Dumbo-MPC: Efficient Fully Asynchronous MPC with Optimal Resilience

Yuan Su[*]
Xi'an Jiaotong University
suyuan@stu.xjtu.edu.cn

Yuan Lu[*]
Institute of Software CAS
luyuan@iscas.ac.cn

Jiliang Li[†]
Xi'an Jiaotong University
jiliang.li@xjtu.edu.cn

Yuyi Wang
CRRC Zhuzhou Institute
yuyiwang920@gmail.com

Chengyi Dong
Xi'an Jiaotong University
2196113533@xjtu.stu.edu.cn

Qiang Tang
The University of Sydney
qiang.tang@sydney.edu.au

Abstract

Fully asynchronous multi-party computation (AMPC) has superior robustness in realizing privacy and guaranteed output delivery (G.O.D.) against asynchronous adversaries that can arbitrarily delay communications. However, none of these protocols are truly practical, as they either have sub-optimal resilience, incur cumbersome communication cost, or suffer from an online phase with extra cryptographic overhead. The only attempting implementation—HoneyBadgerMPC (hbMPC)—merely ensures G.O.D. in some implausible optimistic cases due to a non-robust offline pre-processing phase.

We propose Dumbo-MPC a concretely efficient AMPC-as-a-service design with all-phase G.O.D. and optimal resilience against $t < n/3$ malicious parties (where n is the total number of parties). Similar to hbMPC, Dumbo-MPC has a robust (almost) information-theoretic online phase that can efficiently perform online computations, given pre-processed multiplication triples. To achieve all-phase G.O.D., we design a novel dual-mode offline protocol that can robustly pre-process multiplication triples in asynchrony. The offline phase features $O(n)$ per-triple communication in the optimistic case, followed by a fully asynchronous fallback to a pessimistic path to securely restore G.O.D. in the bad case. To (concretely) efficiently implement the pessimistic path, we devise a concretely efficient zk-proof for the product relationship of secret shares over compact KZG polynomial commitments, which enables us to reduce the degree of two secret shares' product from $2t$ to t and could be of independent interest.

We also implement and extensively evaluate Dumbo-MPC (particularly its offline phase) in varying network settings with up to 31 AWS servers. To our knowledge, we provide the first AMPC implementation with all-phase G.O.D. A recent asynchronous triple generation protocol from Groth and Shoup (GS23) is also implemented and experimentally compared. When $n = 31$, Dumbo-MPC generates 94 triples/sec (almost twice as many as GS23) in the pessimistic case and 349 triples/sec (about 6X of GS23) in the good case.

---
[*] Yuan Su and Yuan Lu contributed equally.
[†] Jiliang Li is the corresponding author.

1 Introduction

The paradigm of multi-party computation (MPC) as a service (MPCaaS) has recently gained significant interests as a promising approach to privacy-preserving distributed systems. Particularly, it is often seen as an enticing solution to overcome the challenge of privacy breaches in blockchains, thereby enabling private smart contracts [16, 85], anonymous broadcasts for transaction diffusion [5, 66], and numerous other tailored decentralized applications [69, 70].

Need for robust MPC in full asynchrony. In many aforementioned scenarios, ensuring G.O.D. is crucial for maintaining the availability of services, particularly in mission-critical applications where timely response is essential. Consider the case of MPCaaS-enabled private smart contracts [16, 85]: clients rely on MPCaaS to evaluate their transactions privately and deliver execution results based on contract clauses. If MPCaaS fails to ensure G.O.D., an adversary could forever block transaction execution, posing a severe denial-of-service threat that could completely censor the private smart contracts.

Despite the urgent demand of G.O.D., most practical MPC implementations [7, 36, 38, 63, 82] fail to provide such necessity [66]. For many additive secret sharing based MPC that tolerate $n-1$ malicious parties [38, 40, 82], it is inherently impossible to realize G.O.D. Even for many MPC protocols [7, 37, 39, 51, 53, 58] that are expected to be robust, their output delivery is conditioned on stringent network synchrony, and unsurprisingly, when they are deployed in an asynchronous network where messages can be arbitrarily delayed (e.g., MPCaaS servers experience unexpected communication interruption due to Internet glitches), their G.O.D. could be violated. Moreover, many MPC protocols [11, 39, 51, 53, 58] even suffer from privacy leakage in a fully asynchronous network, because they heavily rely on the strong assumption of network synchrony to "eject" suspiciously malicious nodes that are temporally unresponsive. As such, they might incorrectly eject honest parties in asynchrony, thereby eventually allowing malicious nodes learn all private inputs.

Given that, it becomes essential to consider more efficient

designs of asynchronous MPC (AMPC), thus ensuring both privacy and G.O.D. in the unstable or even adversarial Internet environment to accommodate mission-critical applications.

Table 1: **Comparison of typical AMPC protocols**

Protocols	Thld. ($t <$)	I.T. online	asyn. G.O.D.	comm. / gate		# secrets to be shared per triple
				Good	Bad	
CHP13 [30]	$n/4$	✓	✓	$O(n)$	$O(n)$	48
CP17 [32]	$n/4$	✓	✓	$O(n)$	$O(n)$	48
DXKR23 [41]‡	$n/3$	✓	✗	$O(n^3)$	–	$2n$
	$n/4$	✓	✓	$O(n^3)$	$O(n^3)$	$2n$
hbMPC [66]	$n/3$	✓	✗	$O(n)$	–	12
HNP08 [60]	$n/3$	✗	✓	$O(n^2)$	$O(n^2)$	-
CP15 [31] †	$n/3$	✗	✓	$O(n)$	$O(n)$	-
CHL21 [28]	$n/3$	✗	✓	$O(n^2)$	$O(n^2)$	-
PCR08 [60]	$n/3$	✓	✓	$O(n^5)$	$O(n^5)$	$9n+6$
CP23 [33]	$n/3$	✓	✓	$O(n^4)$	$O(n^4)$	$6n$
SS23 [77]	$n/3$	✓	✓	$O(n^2)$	$O(n^3)$	$6n$
GS23 [54]	$n/3$	✓	✓	$O(n^2)$	$O(n^3)$	$3n$
GLS24[52]	$n/3$	✓	✓	$O(n)$	$O(n)$	$\sim 230K$
Dumbo-MPC	$n/3$	✓	✓	$O(n)$	$O(n^2)$	$n+6$

† CP15 performs threshold decryption of somewhat homomorphic encryption for each multiplication in online, which might cause serious efficiency issue.
‡ DXKR23 gives a random double-sharing protocol to generate t-degree and $2t$-degree shares of the same randomness, which requires sharing $2n$ secrets to prepare a pair of random double-sharing. However, while using double shares for evaluating multiplication gates during the online phase in asynchrony with $n/3$ corruptions, the $2t$-degree secret shares might fail in reconstructing (as decoding might fail). Only if a lower non-optimal resilience (e.g. $n/4$) is allowed, the robustness of online evaluation can be restored.

1.1 Practical Obstacles of AMPC

Nevertheless, it is challenging to design efficient AMPC while preserving optimal $t < n/3$ resilience,[1] as one cannot distinguish a malicious party that sends nothing and an honest party whose messages are delayed due to network asynchrony. So the protocol has to proceed once receiving $n-t$ distinct parties' messages (which might only solicit $n-2t$ honest parties' messages and therefore omit the rest t honest parties).

Practicality issues of earlier theoretic studies. Despite that AMPC has been studied for more than 30 years, existing results are mostly theoretical, and essentially none of them was ever implemented for various efficiency issues. AMPC was initially studied in the unconditionally-secure setting [13, 14, 75, 78], but these early results have tremendous communication cost (at least n^4 overhead per gate). Some recent results with unconditional security reduce the asymptotic overhead to $O(n^2)$ or $O(n)$ per gate, but most of them [30, 32, 73] only tolerate $t < n/4$ malicious parties. The only unconditionally secure design that realizes linear per-gate communication with optimal $t < n/3$ resilience is a very recent study [52], but has a prohibitive $O(n^{14})$ circuit-independent communication overhead as well as inferior concrete efficiency (requiring thousands of secrets to be verifiably shared per triple).

In the cryptographic setting, the study of AMPC was initialized by Hirt et al. [59, 60], followed by a few theoretical improvements [31, 34] using somewhat/fully homomorphic encryption. Choudhury and Patra [31] realized linear per-gate communication overhead but have costly (n,t) threshold decryption of somewhat homomorphic encryption in its online phase (for multiplication gates). Coretti et al. [35] proposed an asynchronous version of 'BMR' [9] with constant rounds, but it has a costly process using another general-purpose AMPC to pre-compute distributed garbled circuits. Cohen [34] adopted threshold fully homomorphic encryption (tFHE) to get another constant-round AMPC, but it has undesired online computational cost due to expensive FHE evaluations.[2]

hbMPC: trading robustness for efficiency. Recently, HoneyBadgerMPC (hbMPC) [66] gives the first potentially practical (nearly-)asynchronous MPCaaS protocol. The work of hbMPC focuses on optimizing a robust online phase based on Shamir secret sharing [6, 8, 12] with $n/3$ resilience in the pre-processing model, with the price of an adapted *non-robust* offline phase [11, 39] to pre-process the needed multiplication triples. However, the overall robustness of hbMPC is impaired: when its non-robust offline phase is under network attacks, the pre-processing could be stalled; and subsequently, its online phase might also grind to a halt, as a result of forever waiting for the replenishment of multiplication triples. That said, the robustness of hbMPC still relies on a strong assumption that all n parties are honest to cooperatively make the offline phase progress. Unfortunately, such strong assumption could be elusive, especially in adversarial deployment environments like the open Internet.

GS23: a robust offline but with larger per-gate overhead. Very recently, Groth and Shoup (GS23) [54] [3] made one step towards robust asynchronous triple generation by introducing an optimized asynchronous version of 'BGW' [12, 50] in the cryptographic setting. From a high level, it lets each party P_i invoke t-degree asynchronous complete (verifiable) secret sharing (ACSS) to share two secrets a_i and b_i, and adds shares from distinct parties to obtain two random shares $[\![a]\!]$ and $[\![b]\!]$. Then, every party computes $[\![ab]\!]_{2t} = [\![a]\!][\![b]\!]$ as $2t$-degree share of ab, and invokes t-degree ACSS again to re-share $[\![ab]\!]_{2t}$. When instantiating ACSS from (generalized) Pedersen's polynomial commitment, GS23 can let the final re-sharing of $[\![ab]\!]_{2t}$ also carry some zk-proof for the product relationship over Pedersen commitments, thus ensuring everyone to recover a correct t-degree share $[\![ab]\!]$ to complete triple generation. However, in GS23, each party verifiably shares 3 secrets per triple (two for a_i and b_i, and one for $[\![ab]\!]_{2t}$),

---

[1] Note that the optimal resilience of asynchronous Byzantine agreement is $n/3$ [23], implying the same upper bound of resilience of robust AMPCs.

[2] Note that another seemingly viable idea of naively using (n,t) threshold SHE [24, 25, 47] for Shamir shares to pre-process triples (i.e. reflecting an asynchronous version of SPDZ [38, 40, 65]) can also be impractical, as threshold SHE for Shamir shares might have prohibitively large security parameters [20, 81], unless there are sufficient pre-shared noises (whose robust distributed generation in asynchrony is another challenging problem).

[3] Groth and Shoup released the first version of their paper in 2022, followed by a major update in 2023 that included batch Pedersen commitments.

which not only incurs concretely inferior performance, but also causes asymptotically large communication and computational cost, e.g., the per-triple communication is $O(n^2)$ in the good case and even $O(n^3)$ in the bad case.[4]

Can we harvest both robustness and efficiency in AMPC? On the one hand, the almost information-theoretic (I.T.) online phase of hbMPC is promisingly performant, capable of evaluating 8,000+ multiplication gates per second according to our evaluations in the pre-processing model (see Appendix I of the full version [79] for test results), but it suffers from a non-robust offline phase, resulting in a major vulnerability in a fully asynchronous network. On the other hand, robust asynchronous offline protocols like GS23 offer superior robustness but are substantially less efficient, especially when hbMPC might optimistically execute in the good case, GS23 attains a throughput about only 1/6 of hbMPC.

Facing this robustness-efficiency trade-off in state-of-the-art AMPC protocols, [5] we are asking the following question:

Can we push fully asynchronous MPC protocols
with guaranteed output delivery closer to practice
(i.e., as robust as GS23 while as efficient as hbMPC)?

1.2 Our Contribution

We answer the question affirmatively by designing a set of robust and concretely efficient AMPC protocols in the classic online-offline paradigm, called Dumbo-MPC, with amortized linear (resp. quadratic) per-gate communication in the good case (resp. the bad case). To achieve all-phase G.O.D. with optimal resilience, Dumbo-MPC introduces a throughput-optimized fully asynchronous offline protocol to robustly and efficiently pre-process multiplication triples in a batched manner. Dumbo-MPC also inherits hbMPC's almost I.T. online phase, which is robust and already promisingly performant in the pre-processing model, involving only Shamir secret sharing for crucial online efficiency (except for an asynchronous Byzantine agreement component that we also dedicatedly optimize). In sum, our contributions are:

- **New "hidden evaluation" interface of compact KZG polynomial commitment for conveniently proving product relation of committed secret shares.** To apply compact KZG polynomial commitment [62] for reducing the worst-case complexity of GS23, we first devise a new zero-knowledge proof scheme for proving the product relationship of secret shares committed to a triple of KZG commitments. For the purpose, we introduce a new "hidden evaluation" interface and properties to KZG scheme, allowing a party to convert the binding of secret shares from KZG polynomial commitment

into Pedersen commitment. As such, it reduces the problem of proving product relationship over KZG commitments to the very standard problem of proving that over Pedersen commitments. When applying our technique to instantiate Dumbo-MPC using more compact KZG polynomial commitment, we reduce the worst-case communication complexity from GS23's $O(n^3)$ per-gate to $O(n^2)$. Moreover, our approach of proving the product relation of Shamir secret shares over KZG commitments could be of independent interest, e.g., robust publicly auditable/accountable MPCs [61, 76].

- **Robust fully asynchronous offline protocol that can batchedly pre-process multiplication triples more efficiently.** In addition to our asymptotic improvement of the worst-case complexity of GS23, we further enhance concrete efficiency by reducing the number of secrets to be shared by an additional factor of 3, through a sub-protocol that extracts random shares using batching for efficiency. Our approach not only patches the concurrent composability of the state-of-the-art KZG-based batch ACSS [80], but also leverages KZG's homomorphism to redesign the GS23 protocol to extract $t + 1$ random shares from $n - t$ shared secrets via hyper-invertible matrix. More importantly, this still maintains the crucial convenience of proving product relation of secret shares for valid triples, as we can compute the linear combinations of KZG polynomial commitments to bind all extracted random shares. Thereby, our triple generation only verifiably shares n secrets per triple, as opposed to that GS23 requires $3n$.[6]

- **Optimized fast path together with efficient fallback to harvest performance while still preserving G.O.D.** To harvest efficiency from optimistic conditions when no party misbehaves, we add a fast path optimized from hbMPC and earlier studies [11, 39, 66] to generate triples with amortized $O(n)$ communication overhead in the good case. To preserve G.O.D., we design a fallback mechanism enabling honest parties switch to our robust offline protocol when the fast path fails. The fallback is non-trivial, as a naive attempt (directly running the robust protocol after the fast path fails) might cause incorrect online multiplications due to network asynchrony (e.g., for a multiplication gate, someones use a triple from the fast path, but the others use another triple from the robust protocol). We efficiently resolve the threat using a conceptually minimal asynchronous Byzantine agreement for a single bit. Moreover, our fast path saves 3 communication rounds compared to hbMPC, as a bonus of our fallback mechanism eliminating the need of broadcasts to cross-check the consistency of generated triples.

---

[4]Though the state-of-the-art batched ACSS [80] attains $O(n)$ amortized communication cost per secret, its instantiation from Pedersen commitment like GS23 still incurs $O(n^2)$ amortized communication in the bad case.

[5]Besides closely related studies discussed in Introduction, we also review else relevant studies and discuss their limitations in Appendix A.

[6]Our triple generation not only is more efficient than GS23, but also could be *concretely* more efficient than many theoretic designs [33, 52] with similar or even better asymptotic complexities, see Appendix A for such comparisons.

- **The first implementation of AMPC with all-phase G.O.D. and extensive evaluations.** Combining with dedicated optimization of asynchronous consensus component for good-case latency, we then implemented a prototype of Dumbo-MPC. For fair comparison, we also implemented GS23 as a by-product. To our knowledge, we provide the *first* implementation of AMPC with all-phase G.O.D., which is also open-sourced.[7] Extensive evaluations were conducted to demonstrate the performance of Dumbo-MPC under varying network conditions, such as LAN and WAN settings, involving up to 31 AWS EC2 nodes.[8] As a highlight of our experimental results, Table 2 summarizes the pre-processing latency of GS23, hbMPC and Dumbo-MPC in a LAN setting for a task of private Vickrey auction with 100 bidders. Specifically, when $n = 31$, GS23 requires more than 13 minutes to pre-process all 44,571 multiplication triples needed by the Vickrey auction, and hbMPC might even have an infinite latency. In contrast, our pessimistic offline path takes less than 8 minutes, reducing the pre-processing latency by up to 41% compared to GS23.

Table 2: **Pre-processing latency (sec.) for private Vickrey auction with 100 bidders (the bad case, in a LAN setting)**

Protocols	Scale $n =$		
	10	22	31
hbMPC [66]	∞	∞	∞
GS23 [54]	193	464	810
Dumbo-MPC	168 (↓ **13%**)	368 (↓ **21%**)	474 (↓ **41%**)

2 Problem Definition & Technique Overview

For the convenience of readers, we enumerate some notations widely used throughout the paper in Table 3.

Table 3: **Notations**

Notation	Description
n	the total number of parties
t	the maximum number of corrupted parties
N	the batch size of secret sharing
B	the batch size of triple generation
(pk_i, sk_i)	the public-secret key pair of P_i
$[\![r]\!], [\![r]\!]_{2t}$	$(n, t+1)$ and $(n, 2t+1)$ Shamir's secret shares of r
$[\![r]\!]^i$	a Shamir secret share of r held by party P_i
$[n]$	short for $[1, \cdots, n]$
M_{ij}	the j-th element of i-th row of matrix M
κ	the bit length of security parameter
$\varepsilon(\cdot)$	negligible function

2.1 Problem: Asynchronous MPCaaS

Security model. We adopt the standard reliable asynchronous authenticated network with setup assumptions. We also consider the MPC-as-a-Service (MPCaaS) setting with malicious clients. Specifically, our model can be formalized as:

Public identities and trusted setup. There are n designated parties $\{P_1, \cdots, P_n\}$ (i.e., servers) that participate in the MPC protocol. There also exists a public key infrastructure such that each P_i gets and only gets its own secret key sk_i and additionally obtains the public keys $\{pk_i\}_{i \in [n]}$ of all parties. In addition, we assume that the common reference string $(g, g^\alpha, \cdots, g^{\alpha^t}, h, h^\alpha, \cdots, h^{\alpha^t}) \in \mathbb{G}^{2t+2}$ of KZG polynomial commitment [62] is honestly generated and published, which can be done through asynchronous distributed protocols [43].

Fully-connected asynchronous p2p network. We consider a standard asynchronous communication network that consists of secure point-to-point (p2p) channels between each pair of parties. Messages sent between honest parties can be arbitrarily delayed by the adversary, but they remain confidential and must eventually deliver without being tampered.

Adversary corrupting $t < n/3$ servers. We consider malicious, static, probabilistic polynomial-time (P.P.T.) bounded adversaries that can fully control up to $t < n/3$ corrupted parties. Here "static" means that the adversary chooses parties to corrupt before the protocol begins. Noticeably, $t < n/3$ is the optimal resilience of AMPC protocols with G.O.D. [23].

Probably malicious clients. We focus on the enticing MPCaaS model, where k clients submit their private inputs for secure computation. An inherent limitation of asynchrony is that up to c honest clients' inputs may be excluded from computation (where c is the number of malicious clients), because waiting for all k inputs might incur infinite time as corrupted clients might not provide any input. W.l.o.g., we assume $n = k$ and $t = c$ (i.e., each server plays another role of client) for presentation simplicity, unless otherwise specified.

Design goals. We aim to achieve concretely efficient AMPC, featuring robust offline and online phases (as illustrated in Figure 1), tailored for the MPCaaS setting with optimal resilience and all-phase G.O.D. At a high level, the offline phase handles complicated but input-independent tasks like triple generation, and is not latency-critical, so we can focus on its throughput optimization through batching techniques; the online phase leverages the offline pre-processing to enable more responsive function evaluation upon receiving users' private inputs, ensuring suitability for latency-critical applications. The two phases can be formulated as follows.

Offline phase. Syntactically, n parties (servers) take system parameters as input, and each party P_i continuously outputs a linearized sequence of random shares $[\![r_1]\!]^i, [\![r_2]\!]^i \cdots$, and another sequence of multiplication triples $([\![a_1]\!]^i, [\![b_1]\!]^i, [\![a_1 b_1]\!]^i)$, $([\![a_2]\!]^i, [\![b_2]\!]^i, [\![a_2 b_2]\!]^i) \cdots$. The phase shall satisfy the following properties with overwhelming probability:

- *Validity*: for any j-th position in the output sequence of random shares, all honest parties must (eventually) hold consistent t-degree secret shares of some r_j; similarly, for any j-th position in the triples' sequence, all honest parties must (eventually) hold consistent t-degree secret shares of some $a_j, b_j, c_j = a_j b_j$.
- *Secrecy*: it is infeasible for any P.P.T. adversary to predict r_j (resp. $a_j, b_j, a_j b_j$) better than guessing, until the first honest node P_i spreads out its corresponding secret share.
- *Pre-processing liveness (Offline robustness)* : all honest parties' output sequences are ever-growing.

<u>Online phase</u>. Given the above robust offline phase that can continuously generate random shares and multiplication triples, we then aim at realizing an online phase with MP-CaaS interfaces to (probably malicious) clients. As such, for any agreed to evaluate function $f(\cdot)$, the phase can confidentially evaluate f and shall satisfy the next properties with overwhelming probability:
- *Privacy*. For any P.P.T. adversary, it learns nothing in addition to the evaluation result of f.
- *Guaranteed output delivery (online robustness)*. All honest parties (servers) eventually output the evaluation of $f(\cdot)$ on at least $n-t$ distinct clients' inputs.

Performance metrics. We are particularly interested in more efficient AMPC protocols. For the purpose, we primarily consider the key efficiency metric of *(amortized) communication complexity*, reflecting the expected number of bits sent by honest parties to generate each random share or multiplication triple. Moreover, we might estimate the concrete performance of AMPC protocols according to the number of secrets to be (verifiably) shared for each generated multiplication triple. Sometimes, the round of communication is also considered as another indicator to estimate protocols' performance.
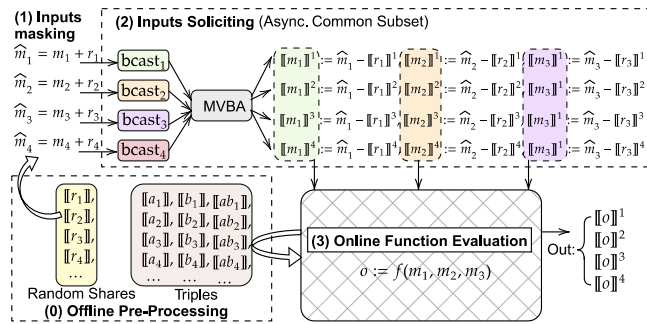

Figure 1: **Offline-online paradigm of AMPCaaS**

2.2 Challenges and Our Techniques

Challenge I: realize both robustness and efficiency while reducing the degree of shares' product. Asynchronous triple generation is at the heart of AMPC. One might immediately realize a "possible" design by adapting the seminal

BGW protocol [12]: (i) All parties first collectively generate some t-degree shares $[\![a]\!]_t$ and $[\![b]\!]_t$ of unbiased randomness a and b, which can be done through a few existing asynchronous protocols [41, 42, 44]; (ii) Everyone locally computes $[\![ab]\!]_{2t} = [\![a]\!]_t [\![b]\!]_t$, which represents $2t$-degree secret share of ab; (iii) Then, each party invokes a "degree reduction" phase by re-sharing $[\![ab]\!]_{2t}$ through another t-degree verifiable secret sharing protocol, such that everyone can interpolate the received shares to obtain $[\![ab]\!]_t$, i.e., reducing the $2t$-degree $[\![ab]\!]_{2t}$ to t-degree $[\![ab]\!]_t$. The attempt appears to be enticing, as one seemingly can plug in the state-of-the-art asynchronous complete secret sharing (ACSS) [3, 80] to instantiate the idea. However, during degree reduction, t malicious parties can re-share arbitrary secret instead of the genuine product, making honest parties fail to decode the correct $[\![ab]\!]_t$ in asynchrony and thus causing the breach of robustness.

Therefore, to guarantee robustness during degree reduction, GS23 introduces batch ACSS (built from Pedersen's polynomial commitments) into the BGW framework [54] and uses zk-proof of product relation over Pedersen commitments to attest the re-sharing of correct $[\![ab]\!]_{2t} = [\![a]\!]_t [\![b]\!]_t$ as inspired by Gennaro et al. [50]. However, Pedersen polynomial commitment is large as broadcasting it already causes $O(n^2)$ bits, and GS23 optimizes this by letting ACSS dealer aggregate a batch of such polynomial commitments (after which, the k-th term in the aggregated commitment is a generalized Pedersen commitment that binds all polynomials' k-th coefficients). The approach has clear drawbacks: (i) it requires a concretely large number of $3n$ secrets to be (verifiably) shared per triple, as every party shares two secrets for generating two random sharings $[\![a]\!]$ and $[\![b]\!]$ and then re-shares $[\![a]\!][\![b]\!]$; (ii) the worst-case communication cost of the resulting batch ACSS is quadratic per secret, incurring cubic overhead per triple.

<u>Our approach</u>. We take advantage of succinct and homomorphic KZG polynomial commitment to realize an alternative batch triple generation protocol improving the enticing approach of GS23, both concretely and asymptotically. This nevertheless requires us to devise a new zk-proof scheme attesting the product relation of a triple of secret shares over KZG commitments. To this end, we introduce the new "hidden evaluation" interface to augment the legacy KZG scheme, enabling each party to compute a Pedersen commitment of a certain evaluation of the polynomial committed to the given KZG commitment. Our new interface can convert the proof-of-product over KZG commitments into the problem of proving that over Pedersen's, and hence, we can realize a more efficient triple generation protocol, as our augmented KZG scheme enables: (i) instantiate an efficient batch ACSS together with our careful patch for concurrent compatibility, thus asymptotically reducing the worst-case communication by another $O(n)$ factor; (ii) employ a more efficient randomness extraction technique via hyper-invertible matrix, reducing the number of shared secrets per triple from $3n$ to n.

Challenge II: fallback from fast path to pessimistic path

might cause incorrect online computation. The non-robust offline protocol adapted from [7, 66] can optimistically generate triples with an amortized $O(n)$ communication overhead. However, when some malicious party starts to misbehave, this fast path protocol might fail to progress. In such bad cases, fallback is needed to restore robustness. But the tricky effect of network asynchrony is that some honest parties might already generate the r-th batch of triples from the fast path, while some other honest parties only obtain the $(r-1)$-th batch of triples from the fast path, which clearly results in a serious vulnerability of disagreed outputs after fallback.

Given this, if the honest nodes trivially quit from unresponsive fast path and then immediately start the pessimistic path, the online computation might no longer be correct, because for a multiplication gate, some honest parties might use triple $([\![a]\!], [\![b]\!], [\![ab]\!])$ generated via fast path but some other parties might use another triple $([\![a']\!], [\![b']\!], [\![a'b']\!])$ generated by the pessimistic path. On the other side, completely withdrawing all fast path outcome after fallback might resolve the inconsistency issue but makes the fast path pointless.

To resolve the above inconsistency issue caused by a failed fast path, one straightforward idea is using an asynchronous common subset (ACS) protocol to handle mode-switching when fallback occurs, because ACS can select the fast path progresses from at least $2f+1$ parties, so the median among the values of ACS output must represent the actual fast fast progress. However, as a heavy variant of asynchronous Byzantine agreement for multiples values, ACS itself is not a cheap primitive and might cause significant performance degradation and subsequently eliminate the efficiency gain from fast path when the fallback is triggered.

Our approach. Since a fast mode-switching module is necessary for a useful dual-mode design, we strive to design a more efficient solution other than naively using ACS, followed by a careful observation on the variance in the fast path progress of honest parties. A key finding is that: for the honest parties, their progresses in the fast path are either the same or have a variance of at most one, when they detect the fast path failed or unresponsive. This is because our fast path requires all parties to be responsive as it requires each party to wait for messages from all parties. If any honest party P_i starts fallback with only obtaining the r-th batch of fast-path triples, then no honest party can obtain the $(r+2)$-th batch of fast-path triples, because P_i would not collaborate in the $(r+2)$-th fast-path triple generation. This hints us at introducing a variant of asynchronous binary agreement for reaching consensus among two consecutive values (tcv-BA) [67]. Namely, every party takes its fast path progress r as input to tcv-ABA during fallback, then tcv-BA returns a common value R, using which, the honest parties can reach a unitary decision on preserving how many fast-path triples before entering the pessimistic path, thereby resolving any inconsistency that might cause incorrect online evaluation.

3 Preliminaries

Asynchronous multi-valued validated Byzantine agreement (MVBA) [2, 4, 26, 41, 46, 68, 83] is a variant of asynchronous Byzantine agreement with output satisfying a public boolean predicate. In particular, MVBA running among n parties is parameterized by a global predicate $Q : \mathbb{X} \times \mathbb{S} \to \{True, False\}$. Here $\mathbb{X}$ represents the domain of inputs and $\mathbb{S}$ denotes the domain of each party's internal states, and Q shall be (i) _monotonic_, i.e., it cannot switch from true to false as the honest party's state S_i evolves, and (ii) _eventually unanimous_, i.e., if $Q(x, S_i)$ is true due to some honest party P_i's state S_i, then eventually $Q(x, S_j)$ becomes true for every honest party P_j's internal state S_j. Such MVBA protocol satisfies the next properties with overwhelming probability: (i) _Termination_, if every honest node P_i starts the protocol with input x_i and internal states S_i s.t. $Q(x_i, S_i) = True$, all honest nodes would eventually output; (ii) _Agreement_, any two honest nodes P_i and P_j output y_i and y_j, respectively, then $y_i = y_j$; (iii) _External Validity_, every honest node P_i's output y_i will be (eventually) valid due to the predicate Q and P_i's internal states S_i.

Asynchronous two-consecutive-value Byzantine agreement (tcv-BA) [67] is an extended asynchronous binary Byzantine agreement where honest parties input two consecutive integers or the same integer. If all honest parties activate a tcv-BA protocol by inputting a value in $\{v, v+1\}$ where $v \in \mathbb{N}$, then the following properties would hold with overwhelming probability: (i) _Termination_, all honest parties would output some value; (ii) _Agreement_, any two honest parties' outputs are the same; (iii) _Validity_, if some honest parties output x, then at least one honest party inputs x.

4 Proof of Product Relation over KZG Polynomial Commitments

As aforementioned, we focus on improving the efficiency of the enticing GS23 protocol through succinct and homomorphic KZG polynomial commitment. This requires us to devise a zero-knowledge proof (zk-proof) scheme for the product relationship of certain polynomial evaluations over KZG commitments, ensuring that any malicious party either re-shares the correct product of $[\![ab]\!]^i_{2t} = [\![a]\!]^i \cdot [\![b]\!]^i$ or does nothing harmful. This section will elaborate on our approach to realizing such efficient zk-proof over KZG commitments.

Adding hidden evaluation interfaces. To prove the product relation of secret shares committed to KZG polynomial commitments, we first introduce new algorithmic interfaces to the KZG scheme. Informally, these new interfaces enable the following: given a KZG polynomial commitment, a polynomial evaluation, and the valid evaluation proof, one can compute the Pedersen commitment of this evaluation and bind this Pedersen commitment to the given KZG commitment.

As illustrated in Figure 2, our construction of hidden evalu-

ation stems from the observation that the VerifyEval function of the KZG scheme only requires a couple of Pedersen-style commitments, $g^{\phi(i)}h^{\hat{\phi}(i)}$ and $g^{\psi(\alpha)}h^{\hat{\psi}(\alpha)}$, to check the equality of two pairings, which hints at that they could be the expected hidden evaluation and corresponding proof. We then prove that our proposed augmented KZG scheme with hidden evaluation interfaces satisfies the properties of (i) *correctness*, (ii) *polynomial binding*, (iii) *evaluation binding*, (iv) *hiding* and (v) *hidden evaluation's binding*, assuming the hardness of discrete logarithm and t-strong Diffie-Hellman problems. We formalize these properties below.

Correctness: Given any $SP \leftarrow \mathsf{Setup}(1^\kappa, t)$ and any $\phi(\cdot) \in \mathbb{Z}_q[x]$, if $C \leftarrow \mathsf{PolyCom}(SP, \phi(\cdot))$, then:
- For any output of $\mathsf{ProveEval}(SP, i, \phi(\cdot), \hat{\phi}(\cdot))$, it can always pass the verification of VerifyEval.
- For any output of $\mathsf{HiddenEval}(SP, i, w_i, \phi(i), \hat{\phi}(i))$, it can always be accepted by VerifyHiddenEval.
- For any output of $\mathsf{HiddenEval}(SP, i, w_i, \phi(i), \hat{\phi}(i))$, there is $\mathsf{OpenHiddenEval}(SP, \phi(i), \hat{\phi}(i), T_i) = 1$.

Evaluation binding: For any P.P.T. adversary $\mathcal{A}$,

$$\Pr\left[\begin{array}{c} SP \leftarrow \mathsf{Setup}(1^\kappa, t); \\ (C, (i, \phi(i), \hat{\phi}(i), w_i), (i, \phi(i)', \hat{\phi}(i)', w_i')) \leftarrow \mathcal{A}(SP): \\ \mathsf{VerifyEval}(SP, C, i, \phi(i), \hat{\phi}(i), w_i) = 1 \wedge \\ \mathsf{VerifyEval}(SP, C, i, \phi(i)', \hat{\phi}(i)', w_i') = 1 \wedge \phi(i) \neq \phi(i)' \end{array}\right] \leq \varepsilon(\kappa)$$

Polynomial binding: For any P.P.T. adversary $\mathcal{A}$,

$$\Pr\left[\begin{array}{c} SP \leftarrow \mathsf{Setup}(1^\kappa, t); \\ (C, I_1, I_2, \{(\phi(i), \hat{\phi}(i), w_i)\}_{i \in I_1 \cup I_2}) \leftarrow \mathcal{A}(SP) \\ \text{where } I_1 \subset [n], I_2 \subset [n], \text{and } |I_1| = |I_2| = t+1: \\ \forall i \in I_1 \cup I_2, \mathsf{VerifyEval}(SP, C, i, \phi(i), \hat{\phi}(i), w_i) = 1 \wedge \\ (\phi_1(\cdot), \hat{\phi}_1(\cdot)) \leftarrow Interpolate(I_1, \{\phi(i), \hat{\phi}(i)\}_{i \in I_1}) \wedge \\ (\phi_2(\cdot), \hat{\phi}_2(\cdot)) \leftarrow Interpolate(I_2, \{\phi(i), \hat{\phi}(i)\}_{i \in I_2}) \wedge \\ \phi_1(\cdot) \neq \phi_2(\cdot) \end{array}\right] \leq \varepsilon(\kappa)$$

(Augmented) hiding: The polynomial commitment and all hidden evaluations should not reveal any additional information about the polynomial beyond t shares that are already known to the adversary. This is an enhanced hiding property of the original KZG scheme, which now also accounts for outputs from the hidden evaluation interface. Formally, for any P.P.T. adversary, a P.P.T. simulator $(\mathsf{Sim}_0, \mathsf{Sim}_1, \mathsf{Sim}_2, \mathsf{Sim}_3)$ exists, such that the next two distributions are identical:

Real world:
$$\left\{\begin{array}{l} SP \leftarrow \mathsf{Setup}(1^\kappa, t); (\phi(\cdot), I_1, I_2) \leftarrow \mathcal{A}(SP); \\ (C, \hat{\phi}(\cdot)) \leftarrow \mathsf{PolyCom}(SP, \phi(\cdot)); \\ \forall i \in I_1, (\phi(i), \hat{\phi}(i), w_i) \leftarrow \mathsf{ProveEval}(SP, i, \phi(\cdot), \hat{\phi}(\cdot)); \\ \forall i \in I_2, (T_i, \omega_i) \leftarrow \mathsf{HiddenEval}(SP, i, \phi(\cdot), \hat{\phi}(\cdot)) \\ : (C, \{i, \phi(i), \hat{\phi}(i), w_i\}_{i \in I_1}, \{i, T_i, \omega_i\}_{i \in I_2}) \text{ where } |I_1| \leq t \end{array}\right\}$$

Ideal world:
$$\left\{\begin{array}{l} (SP, st) \leftarrow \mathsf{Sim}_0(1^\kappa, t); (\phi(\cdot), I_1, I_2) \leftarrow \mathcal{A}(SP); \\ (C, c) \leftarrow \mathsf{Sim}_1(st); \\ (\{w_i\}_{i \in I_1}, \phi'(\cdot), \hat{\phi}'(\cdot)) \leftarrow \mathsf{Sim}_2(st, (C, c), \{i, \phi(i)\}_{i \in I_1}); \\ \{T_i, \omega_i\}_{i \in I_2} \leftarrow \mathsf{Sim}_3(st, (C, c), \phi'(\cdot), \hat{\phi}'(\cdot), I_2) \\ : (C, \{i, \phi(i), \hat{\phi}(i), w_i\}_{i \in I_1}, \{i, T_i, \omega_i\}_{i \in I_2}) \text{ where } |I_1| \leq t \end{array}\right\}$$

Hidden evaluation's binding: For all adversaries $\mathcal{A}$,

$$\Pr\left[\begin{array}{c} SP \leftarrow \mathsf{Setup}(1^\kappa, t); \\ (C, i, (T_i, \omega_i, \phi'(i), \hat{\phi}'(i)), (\phi(i), \hat{\phi}(i), w_i)) \leftarrow \mathcal{A}(SP): \\ \mathsf{VerifyHiddenEval}(SP, C, i, T_i, \omega_i) = 1 \wedge \\ \mathsf{OpenHiddenEval}(SP, \phi'(i), \hat{\phi}'(i), T_i) = 1 \wedge \\ \mathsf{VerifyEval}(SP, C, i, \phi(i), \hat{\phi}(i), w_i) = 1 \wedge \hat{\phi}'(i) \neq \hat{\phi}(i) \end{array}\right] \leq \varepsilon(\kappa)$$

REMARKS. Clearly, our augmented KZG polynomial commitment with hidden evaluation (KZG PCwHE) satisfies all properties of the original KZG scheme [62]. Additionally, it provides a convenient interface to convert the binding of a secret share from a given KZG commitment to another Pedersen commitment that not only binds the secret share originally committed to KZG but also leaks nothing about the secret. For detailed proofs of KZG PCwHE, see Appendix B of the full version [79].

KZG Polynomial Commitment w/ Hidden Evaluation

$\mathsf{Setup}(1^\kappa, t)$: generate bilinear pairing group $\mathcal{G} = (e, \mathbb{G}, \mathbb{G}_T)$ and randomly sample $\alpha, \tau \in \mathbb{Z}_q$. Let g be a random generator of $\mathbb{G}$, set $h = g^\tau$ and return $SP = \{\mathcal{G}, \{g^{\alpha^i}, h^{\alpha^i}\}_{i=0}^t\}$.

$\mathsf{PolyCom}(SP, \phi(\cdot))$: sample random t-degree polynomial $\hat{\phi}(\cdot)$, and compute $C = g^{\phi(\alpha)}h^{\hat{\phi}(\alpha)}$.

$\mathsf{ProveEval}(SP, i, \phi(\cdot), \hat{\phi}(\cdot))$: compute $w_i = g^{\psi_i(\alpha)}h^{\hat{\psi}_i(\alpha)}$, where $\psi_i(x) = \frac{\phi(x)-\phi(i)}{x-i}$ and $\hat{\psi}_i(x) = \frac{\hat{\phi}(x)-\hat{\phi}(i)}{x-i}$, output $(i, \phi(i), \hat{\phi}(i), w_i)$.

$\mathsf{VerifyEval}(SP, C, i, \phi(i), \hat{\phi}(i), w_i)$: output 1 if $e(C/(g^{\phi(i)}h^{\hat{\phi}(i)}), g) = e(w_i, g^\alpha/g^i)$, output 0 otherwise.

———————— Below are hidden evaluation interfaces ————————

$\mathsf{HiddenEval}(SP, i, w_i, \phi(i), \hat{\phi}(i))$: compute $T_i = g^{\phi(i)}h^{\hat{\phi}(i)}$ and output (i, T_i, ω_i) where $\omega_i = w_i$.

$\mathsf{OpenHiddenEval}(SP, \phi(i), \hat{\phi}(i), T_i)$: if $T_i = g^{\phi(i)}h^{\hat{\phi}(i)}$, output 1, otherwise output 0.

$\mathsf{VerifyHiddenEval}(SP, C, i, T_i, \omega_i)$: if $e(C/T_i, g) = e(\omega_i, g^\alpha/g^i)$, output 1, otherwise output 0.

NOTE: we also use BatchVerifyEval and BatchVerifyHiddenEval to denote the batch versions of VerifyEval and VerifyHiddenEval, which simultaneously perform many verifications for efficiency.

Figure 2: **Our augmented KZG polynomial commitment with new hidden evaluation interfaces (KZG PCwHE).**

Proving the product relation over KZG commitments. Given our augmented KZG scheme with hidden evaluation interfaces, we realize a convenient way to prove the product relationship of secret shares over a triple of (publicly known) KZG commitments, as illustrated in Figure 3. Particularly, the proposed proof-of-knowledge (PoK) zk-proof attests the following statement:

$\mathbf{PoK_{Prod\_KZG}}[\{(\phi_x(i_x), \hat{\phi}_x(i_x), w_x)\}_{x \in \{a,b,c\}}:$

$\phi_a(i_a) \cdot \phi_b(i_b) = \phi_c(i_c) \wedge$

$\mathsf{VerifyEval}(SP, C_x, i_x, \phi_x(i_x), \hat{\phi}_x(i_x), w_x) = 1, \forall x \in \{a, b, c\}]$

which proves that the secret share $\phi_c(i_c)$ committed to KZG commitment C_c is indeed a correct product of two legitimate

<div style="border: 1px solid black;">

PoK of Product over KZG Commitments PoK$_{\mathsf{Prod\_KZG}}$

The prover $\mathcal{P}$ and the verifier $\mathcal{V}$ both know the public statement $(SP, C_a, C_b, C_c, i_a, i_b, i_c)$ where $\{C_x\}_{x \in \{a,b,c\}}$ are KZG polynomial commitments and $\{i_x\}_{x \in \{a,b,c\}}$ are points to evaluate polynomials. $\mathcal{P}$ has private witness $\{(\phi_x(i_x), \hat{\phi}_x(i_x), w_x)\}_{x \in \{a,b,c\}}$, s.t. $\forall x \in \{a, b, c\}$, VerifyEval$(SP, C_x, i_x, \phi_x(i_x), \hat{\phi}_x(i_x), w_x) = 1$.

// Prover $\mathcal{P}$

- PoK$_{\mathsf{Prod\_KZG}}.\mathcal{P}(\{(C_x, i_x, \phi_x(i_x), \hat{\phi}_x(i_x), w_x)\}_{x \in \{a,b,c\}})$:

(a) compute HiddenEval$(SP, i_x, w_x, \phi(i_x), \hat{\phi}(i_x))$ to obtain (T_x, ω_x) for each $x \in \{a, b, c\}$;

(b) invoke the prover of the proof-of-product scheme for values committed to Pedersen commitments (see Figure 13 in the full version [79]) with taking $\{(\phi_x(i_x), \hat{\phi}_x(i_x))\}_{x \in \{a,b,c\}}$ as witness, and obtain the proof π regarding the product relation of (T_a, T_b, T_c).

(c) output proof $= (\{(T_x, \omega_x)\}_{x \in \{a,b,c\}}, \pi)$.

// Verifier $\mathcal{V}$

- PoK$_{\mathsf{Prod\_KZG}}.\mathcal{V}(\{(C_x, i_x)\}_{x \in \{a,b,c\}}, \mathsf{proof})$:

(a) parse proof as $(\{(T_x, \omega_x)\}_{x \in \{a,b,c\}}, \pi)$;

(b) verify VerifyHiddenEval$(SP, C_x, i_x, T_x, \omega_x)=1$ for $x \in \{a, b, c\}$;

(c) verify the proof-of-product π regarding (T_a, T_b, T_c) (see Figure 13 in the full version [79]), return 1 if all checks pass and 0 otherwise.

</div>

Figure 3: **Zk-proof of product relation for a triple of evaluations of polynomials committed to KZG commitments.**

secrets' shares $\phi_a(i_a)$ and $\phi_b(i_b)$ that are respectively committed to two KZG commitments C_a and C_b. Intuitively, the proof scheme first computes hidden evaluation as a Pedersen commitment for each secret share, and then proves the product relationship over these derived Pedersen commitments. In the next section when we instantiate our asynchronous triple generation protocol, this zk-proof plays a crucial role in preventing malicious parties from re-sharing arbitrary secrets instead of the genuine product of two expected shares.

Algorithm 1 Asynchronous random share generation ▷ Code of P_i

Input: N the batch size of BACSS, and other system parameters

Output: rnd_shares $= \{(C_k, [\![r_k]\!], [\![\hat{r}_k]\!], w_k)_{k=1}^{N(t+1)}\}$

    // 1. Invoke BACSS to share randomness

1: $\mathcal{T}_i \leftarrow \emptyset$, shares$_i \leftarrow \emptyset$, rnd_shares $\leftarrow \emptyset$

2: activate BACSS_share$_j$ instance for every $j \in [n]$

3: uniformly sample N secrets $s_{i1}, \cdots, s_{iN}$ from $\mathbb{Z}_q$

4: invoke BACSS_share$_i$ as dealer to share $s_{i1}, \cdots, s_{iN}$

    // Each P_i generate n private-public key pairs for IND-CCA PKE,

    // so the j-th key pair is exclusively used in dealer P_j's BACSS.

    // 2. Select $n-t$ completed BACSS instances

5: **upon** getting $(C_{jk}, [\![s_{jk}]\!]^i, [\![\hat{s}_{jk}]\!]^i, w_{jk}^i)_{k \in [N]}$ from BACSS_share$_j$

6:     $\mathcal{T}_i \leftarrow \mathcal{T}_i \bigcup \{j\}$

7:     shares$_i \leftarrow$ shares$_i \bigcup \{(C_{jk}, [\![s_{jk}]\!]^i, [\![\hat{s}_{jk}]\!]^i, w_{jk}^i)_{k \in [N]}\}$

8:     **if** $|\mathcal{T}_i| = 2t+1$

9:         call MVBA using a snapshot of $\mathcal{T}_i$ as input and $\mathcal{T}_i$ as state

        // MVBA's predicate waits for local $\mathcal{T}_i$ is a superset of input

10: **wait** for MVBA output $\mathcal{T}$, w.l.o.g., let $\mathcal{T} = \{P_1, \cdots, P_{2t+1}\}$

    // 3. Extract random shares

11: **for** $k = 1$ to N

12:     $([\![r_{1k}]\!]^i, \cdots, [\![r_{(t+1)k}]\!]^i) \leftarrow M([\![s_{1k}]\!]^i, \cdots, [\![s_{(2t+1)k}]\!]^i)^T$

13:     $([\![\hat{r}_{1k}]\!]^i, \cdots, [\![\hat{r}_{(t+1)k}]\!]^i) \leftarrow M([\![\hat{s}_{1k}]\!]^i, \cdots, [\![\hat{s}_{(2t+1)k}]\!]^i)^T$

14:     **for** $l = 1$ to $t+1$

15:         $C_{lk} = \prod_{j=1}^{2t+1} (C_{jk})^{M_{lj}}, \quad w_{lk}^i = \prod_{j=1}^{2t+1} (w_{jk}^i)^{M_{lj}}$

16:         rnd_shares $\leftarrow$ rnd_shares $\bigcup \{(C_{lk}, [\![r_{lk}]\!]^i, [\![\hat{r}_{lk}]\!]^i, w_{lk}^i)\}$

17: **return** rnd_shares

---

5 Robust AMPC Offline Protocols

The core of Dumbo-MPC is a robust yet still efficient asynchronous offline phase. Building on our previously devised proof-of-product scheme over KZG commitments, this section will elaborate on our robust AMPC offline protocols.

5.1 Asynchronous Random Share Generation

We start by presenting an asynchronous random share generation (AsyRanShGen) protocol, optimized through various batching techniques for enhanced efficiency, capable of generating t-degree random shares with an amortized communication cost of $O(\kappa n)$. We employ a randomness extraction method utilizing hyper-invertible matrices (where every square sub-matrix is invertible) [11], which was recently adapted by Das et al. for the asynchronous setting [41] though without further batching from batch ACSS.

Patch the concurrent composability for batch ACSS. One might suggest directly integrating the state-of-the-art batch asynchronous complete secret sharing (BACSS) hbACSS [80] into Das et al.'s design to implement a batched AsyRanShGen protocol.[9] Here BACSS allows a dealer to share a batch of secrets $(s_1, \cdots, s_N)$ across the whole network, such that (i) each honest party P_i would output a batch of N secret shares (where the k-th output $[\![s_k]\!]^i$ corresponds to a share of the k-th input secret s_k), or (ii) probably all honest parties eventually output nothing, if the dealer is malicious. However, we identify a flaw of the above seemingly viable idea due to lacking concurrent composability in the original hbACSS, which might even cause realistic secrecy attacks allowing a malicious dealer copy transcripts from another honest dealer and thus learn the secrets (see Appendix C.3 of the full version [79] for the detailed attack).

We therefore make necessary yet still minimal adaptions of hbACSS to patch its concurrent composability: (i) use IND-CCA secure public key encryption (PKE) instead of IND-CPA PKE in the original hbACSS design; (ii) specify that each party wouldn't re-use PKE keys in different dealers' hbACSS instances. Such enhancement enables us to prove the crucial secrecy of concurrent hbACSS instances. Syntactically, we

---

[9]Note that hbACSS [80] is more preferred than another state-of-the-art BACSS protocol Bingo [3] due to significantly better concrete efficiency, as the computation and communication of Bingo are at least 3 times of hbACSS.

also explicitly require that the honest parties in BACSS additionally output N KZG polynomial commitments besides N secret shares, such that each output KZG commitment fixes a polynomial whose evaluations correspond to the k-th output shares of all parties. The communication complexity of this adapted hbACSS still is $O(\kappa n N + \kappa n^2 \log n)$. When letting the batch size sufficiently large such as $N = \Omega(n \log n)$,[10] the amortized communication cost is $O(\kappa n)$ per shared secret. We refrain from representing the whole protocol of the adapted hbACSS, and defer its details to Appendix C.2 of the full version [79].

AsyRanShGen protocol. Algorithm 1 describes the protocol for batchedly generating random shares atop concurrently composable BACSS protocols, which has the following steps:

1. *Share randomness*. Each P_i samples a batch of random secrets, and subsequently starts as a dealer of one BACSS protocol to verifiably share them in a batched manner.

2. *Solicit sufficient (probably biased) randomness shares*. In the asynchronous setting, one cannot wait for the terminations of all parties' BACSS protocols, so once a party P_i outputs in $n - t$ distinct dealers' BACSS protocols, it activates an MVBA by taking the indices of these finished BACSS as input. MVBA ensures that all honest parties can select a common set of $n - t$ BACSS protocols with distinct dealers, and all honest parties can eventually obtain shares from these $n - t$ solicited BACSS protocols.

3. *Extract uniform randomness via hyper-invertible matrix*. Among the $n - t$ selected BACSS protocols, at least $t + 1$ of them shall have honest dealers that indeed share uniform randomness sampled from $\mathbb{Z}_q$, and at most t of them could be chosen by the adversary.

   Thus, to extract as many as possible random shares from these $n - t$ BACSS protocols, we leverage the idea of randomness extraction via Vandermonde matrix $M \in \mathbb{Z}_q^{(t+1) \times (2t+1)}$, where the j-th row is $M_j = (\eta_j^0, \eta_j^1, \cdots, \eta_j^{2t})$ and $\eta_j \in \mathbb{Z}_q$ is distinct for each j. So given $2t + 1$ BACSS protocols (each of which batchedly shares N secrets), the honest parties can extract a number of $(t + 1) * N$ random shares.

   The polynomial commitments and evaluation proofs related to the extracted random shares can also be locally computed by each honest party (without interacting), due to the additive homomorphism of KZG scheme. Later, we will leverage these resulting polynomial commitments for robust triple generation.

Security of AsyRanShGen. The properties of AsyRanShGen can be summarized as the following security theorem.

Theorem 1 (Termination, Consistency, Secrecy and Randomness of AsyRanShGen). *If all honest parties activate the*

AsyRanShGen protocol (running among n parties against up to t malicious corruptions), then the following properties hold with all but negligible probability:

- *Termination. All honest parties would eventually output a set* rnd_shares $= \{(C_k, [\![r_k]\!]^i, [\![\hat{r}_k]\!]^i, w_k^i)\}_{k=1}^{N(t+1)}$.

- *Consistency. The output* rnd_shares *of all honest parties are "consistent", namely: (i) they carry the same* $\{C_k\}_{k=1}^{N(t+1)}$; *(ii) for each* $(C_k, [\![r_k]\!]^i, [\![\hat{r}_k]\!]^i, w_k^i) \in$ rnd_shares, VerifyEval$(SP, C_k, i, [\![r_k]\!]^i, [\![\hat{r}_k]\!]^i, w_k^i) = 1$.

- *Secrecy and randomness. The adversary cannot predicate each* r_k *better than guessing over* $\mathbb{Z}_q$.

We defer the full proof of the above security theorem to Appendix G.1 of the full version [79], as most analysis is rather standard and mirrors the literature [17, 41, 55]. Particularly considering that we already patch the concurrent composability of BACSS, it enables us construct a simulator that always simulates decryption queries even if in face of malicious BACSS dealers, and such simulated decryption would not leak the secrets of other concurrent BACSS instances.

Complexities of AsyRanShGen. In the AsyRanShGen protocol, $\Theta(n)$ instances of BACSS are involved. If each BACSS has a sufficient batch size $N = \Omega(n \log n)$, the communication cost of n BACSS protocols becomes $O(\kappa n^3 \log n)$ in total. Additionally, one MVBA protocol with n-bit input is invoked, and its state-of-the-art instantiation costs $O(\kappa n^3)$ bits [4, 45]. In sum, AsyRanShGen can batchedly generate $N(t + 1)$ random shares with a total communication cost of $O(\kappa n^3 \log n)$ bits, which corresponds to an amortized communication complexity of $O(\kappa n)$ per generated random share.

5.2 Asynchronous Random Triple Generation

Here we are ready to present the protocol of asynchronous random multiplication triple generation (AsyRanTriGen), which can be thought of an asynchronous, optimal-resilient, and batched version of the seminal BGW paradigm and is also concretely and asymptotically more efficient than GS23.

Algorithm 2 Asynchronous mul. triples generation   ▷ Code of P_i

Input: B batch size of triple generation and other system parameters
Output: Beaver_triples $= \{([\![a_k]\!]^i, [\![b_k]\!]^i, [\![c_k]\!]^i)\}_{k=1}^B$

1: $\mathcal{T}_i \leftarrow \emptyset$, prod_shares$_i \leftarrow \emptyset$, Beaver_triples $\leftarrow \emptyset$
   // **1. Generate random shares**
2: activate AsyRanShGen with taking as input $N = (2B/(t+1))$ randomness, and wait for AsyRanShGen returns rnd_shares $= \{(C_k, [\![r_k]\!]^i, [\![\hat{r}_k]\!]^i, w_k^i)\}_{k \in [2B]}$
3: let first half of rnd_shares be $\{(C_{ak}, [\![a_k]\!]^i, [\![\hat{a}_k]\!]^i, w_{ak}^i)\}_{k \in [B]}$ and last half of rnd_shares be $\{(C_{bk}, [\![b_k]\!]^i, [\![\hat{b}_k]\!]^i, w_{bk}^i)\}_{k \in [B]}$
   // **2. Re-share $2t$-degree product**
4: **for** $k = 1$ to B
5:   $[\![c_k]\!]_{2t}^i \leftarrow [\![a_k]\!]^i \cdot [\![b_k]\!]^i$
6:   $(C_{ck}^i, \hat{\phi}_{ck}^i(\cdot)) \leftarrow \text{PolyCom}(SP, \phi_{ck}^i(\cdot))$, s.t. $\phi_{ck}^i(0) = [\![c_k]\!]_{2t}^i$
7:   $w_{ck,0}^i \leftarrow \text{ProveEval}(SP, 0, \phi_{ck}^i(\cdot), \hat{\phi}_{ck}^i(\cdot))$
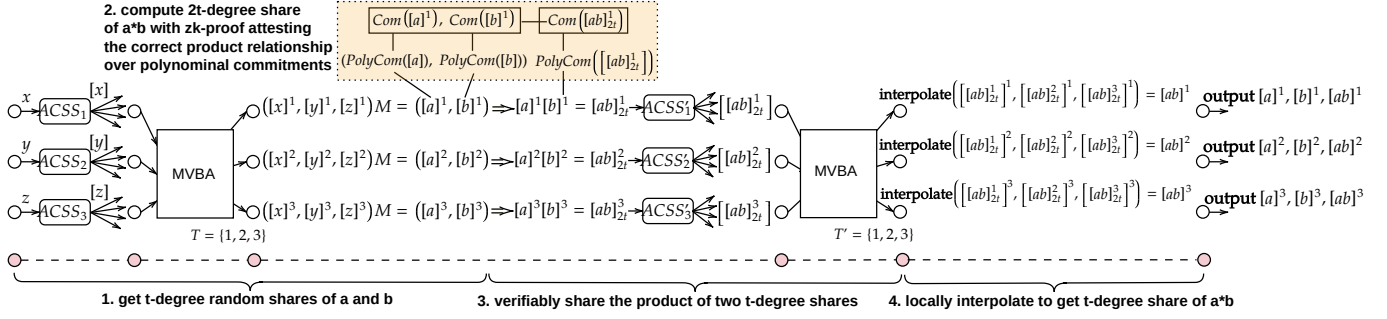

Figure 4: **The overview of asynchronous multiplication triple generation protocol (exemplified without batching).**

8: $\text{proof}_k^i \leftarrow \text{PoK}_{\text{Prod\_KZG}}.\mathcal{P}\left(\left\{\begin{array}{c}(C_{ak},i,[\![a_k]\!]^i,[\![\hat{a}_k]\!]^i,w_{ak}^i),\\(C_{bk},i,[\![b_k]\!]^i,[\![\hat{b}_k]\!]^i,w_{bk}^i),\\(C_{ck}^i,0,\phi_{ck}(0),\hat{\phi}_{ck}(0),w_{ck,0}^i)\end{array}\right\}\right)$

cf. Fig 3 for details of the above proof of product relationship

9: activate $\text{BACSS\_share}_j'$ instance for every $j \in [n]$

10: invoke $\text{BACSS\_share}_i'$ as dealer to re-share $\{[\![c_k]\!]_{2t}^i\}_{k\in[B]}$ using KZG commitments $\{C_k^i\}_{k\in[B]}$ with broadcasting $\{\text{proof}_k^i\}_{k\in[B]}$

// **3. Wait for $n-t$ completed re-shares of product**

11: **upon** receiving output from $\text{BACSS\_share}_j'$ instance:

12: **if** $\text{PoK}_{\text{Prod\_KZG}}.\mathcal{V}\left(\left\{\begin{array}{c}(C_{ak},j),\\(C_{bk},j),\\(C_{ck,0}^j,0)\end{array}\right\}, \text{proof}_k^j\right)=1$ for $\forall\, k \in [B]$:

13: $\quad\quad \text{prod\_shares}_i \leftarrow \text{prod\_shares}_i \bigcup \{([\![[\![c_k]\!]_{2t}^j]\!]^i)_{k\in[B]}\}$

14: $\quad\quad \mathcal{T}_i \leftarrow \mathcal{T}_i \bigcup \{j\}$

15: $\quad\quad$ **if** $|\mathcal{T}_i| = 2t+1$

16: $\quad\quad\quad$ call MVBA' using a snapshot of $\mathcal{T}_i$ as input and $\mathcal{T}_i$ as state

// **4. Locally interpolate t-degree share of product**

17: **wait** for MVBA output $\mathcal{T}$, w.l.o.g., let $\mathcal{T} = \{P_1,\cdots,P_{2t+1}\}$

18: **for** $k = 1$ to B

19: $\quad [\![c_k]\!]^i \leftarrow \sum_{j=1}^{2t+1} \lambda_j [\![[\![c_k]\!]_{2t}^j]\!]^i$, where λ_j represents the evaluation of the j-th Lagrange interpolating polynomial at zero point

20: $\quad \text{Beaver\_triples} \leftarrow \text{Beaver\_triples} \bigcup \{([\![a_k]\!]^i, [\![b_k]\!]^i, [\![c_k]\!]^i)\}$

21: **return** Beaver_triples

AsyRanShGen protocol. As illustrated in Algorithm 2, the protocol proceeds through running the following four steps:

1. *Batchedly generate random shares*. All parties execute the AsyRanShGen protocol presented in the earlier subsection, where each party P_i randomly chooses $N = \frac{2B}{t+1}$ secrets to share via its BACSS instance and would obtain $\{(C_k, [\![s_k]\!]^i, [\![\hat{s}_k]\!]^i, w_k^i)_{k\in[2B]}$, i.e., $2B$ random shares along with all related KZG commitments and evaluations proofs. Presentation-wise, we partition the output set into two halves, denoted as $\{(C_{ak}, [\![a_k]\!]^i, [\![\hat{a}_k]\!]^i, w_{ak}^i)\}_{k\in[B]}$ and $\{(C_{bk}, [\![b_k]\!]^i, [\![\hat{b}_k]\!]^i, w_{bk}^i)\}_{k\in[B]}$, respectively.

2. *Locally compute 2t-degree shares of products*. Given the output obtained from the AsyRanShGen protocol, every party P_i computes $[\![c_k]\!]_{2t}^i = [\![a_k]\!]^i \cdot [\![b_k]\!]^i$ for each $k \in [B]$.

3. *Re-share the correct 2t-degree product*. Then P_i invokes another BACSS protocol as dealer to re-share $[\![c_k]\!]_{2t}^i$ using KZG polynomial commitment C_{ck}^i.

   We also require P_i to compute and broadcast some additional $\{\text{proof}_k^i\}$ attesting the correctness of the re-shared 2t-degree product $[\![c_k]\!]_{2t}^i$. Particularly, each $\text{proof}_k^i = \{T_{ak}^i, \omega_{ak}^i, T_{bk}^i, \omega_{bk}^i, T_{ck,0}^i, \omega_{ck,0}^i, \pi_k^i\}$ is a zero-knowledge proof of the product relationship over KZG commitments C_{ak}, C_{bk} and C_{ck}^i (as earlier described in Figure 3). Here $(T_{ak}^i, \omega_{ak}^i)$ are a hidden evaluation (i.e. a Pedersen commitment) and a hidden evaluation proof regarding KZG commitment C_{ak}, attesting that T_{ak}^i commits the i-th evaluation of the polynomial fixed by C_{ak}; Similarly, ω_{bk}^i attests T_{bk}^i commits an evaluation of the polynomial fixed by C_{bk} at point i; While $\omega_{ck,0}^i$ proves that $T_{ck,0}^i$ commits the 0-point secret of the polynomial committed to C_{ck}^i.

4. *Reconstruct t-degree shares of products to obtain triples*. After everyone re-shares its local 2t-degree shares, each P_i activates another MVBA protocol to decide a common set of $2t+1$ completed re-sharings. Finally, each P_i locally interpolates the t-degree share $[\![c_k]\!]^i$ of the product. As P_i also has shares $[\![a_k]\!]^i$ and $[\![b_k]\!]^i$, so it can return a random triple $([\![a_k]\!]^i, [\![b_k]\!]^i, [\![c_k]\!]^i)$ for each $k \in [B]$.

Security of AsyRanTriGen. The security of AsyRanTriGen can be summarized by the following theorem.

Theorem 2 (Termination, Validity, and Secrecy of AsyRanTriGen)**.** *In an AsyRanTriGen protocol running among n parties with up to $t < n/3$ corruptions, each party P_i outputs a batch of triples $\{([\![a_k]\!]^i, [\![b_k]\!]^i, [\![c_k]\!]^i)\}_{k=1}^B$, and the following properties shall hold with all but negligible probability:*

- *Termination. If all honest parties activate the protocol, all of them would output $\{([\![a_k]\!]^i, [\![b_k]\!]^i, [\![c_k]\!]^i)\}_{k=1}^B$.*

- *Validity. For each $k \in B$ and $x \in \{a,b,c\}$, honest parties can interpolate their output shares $\{[\![x_k]\!]^i\}$ to obtain a unique t-degree polynomial whose zero-point is x_k, and the interpolated $\{x_k\}_{x\in\{a,b,c\}}$ satisfying $a_k \cdot b_k = c_k$.*

- *Secrecy. For any output triple $([\![a_k]\!], [\![b_k]\!], [\![c_k]\!])$, adversary learns nothing about a_k, b_k and c_k except $a_k \cdot b_k = c_k$.*

We provide the intuitions of proofs hereunder, and defer the detailed analysis to Appendix G.2 of the full version [79]:

- For termination, this is trivial, as its violation is reducible to breach either of (i) the correctness of BACSS; (ii) the completeness of BACSS; (iii) the termination of MVBA.

- For validity, the arguments are: (i) the random shares $\llbracket a \rrbracket$ and $\llbracket b \rrbracket$ obtained by honest parties must be consistent to fixed t-degree polynomials, because they are evaluations bounded to the same polynomial commitments; (ii) the honest parties' secret shares $\llbracket c \rrbracket$ must correspond to the same t-degree polynomial with $c = a \cdot b$, because otherwise, either the hidden evaluation's binding of our augmented KZG scheme is violated or the knowledge soundness of the proof of product relationship is breached.

- For secrecy, we construct a P.P.T. simulator using the setup trapdoor of KZG commitment and black-box simulators of product proof and IND-CCA encryption, so we can get into a hybrid world where the adversary interacts with the above simulator, and the adversary's view in this hybrid world is computationally indistinguishable from its view in the real-world execution. Observing that the adversary in the final hybrid world can only information-theoretically get t shares of $\llbracket a \rrbracket$, $\llbracket b \rrbracket$ and $\llbracket c \rrbracket$, secrecy is therefore proven in the real-world execution.

Complexities of AsyRanTriGen. An AsyRanTriGen protocol that generates B triples involves an AsyRanShGen instance generating $2B$ random shares, n BACSS protocol instances (each of which shares B secrets), and another MVBA instance. For sufficiently large batch B, the communication of n BACSS instances with batch size B becomes the dominating factor, resulting in amortized $O(\kappa n^2)$ bits per triple.

6 Adding Fast Path for Triple Generation

Now we elaborate on how to further harvest efficiency from optimistic conditions using a concretely and asymptotically more efficient non-robust fast path, with preserving G.O.D. for all possible bad cases, through a simplistic yet still fully asynchronous fallback mechanism.

Fast path protocol. Algorithm 3 describes a non-robust *optimistic* random triple generation protocol (OptRanTriGen) adapted from [11, 39, 66]. For completeness, we repeat the protocol here with our tailored optimization (which removes $O(n)$ broadcasts and thus saves 3 additional rounds), while deferring its security analysis and the used algorithms of plain Shamir's secret sharing to Appendices G.3 and F of the full version [79], respectively.

In OptRanTriGen, each party shares the same random secret to all parties twice, using t-degree and $2t$-degree plain Shamir secret sharing without verifiability. Then, everyone waits for these shares from all parties in order to extract $t + 1$ random double sharings $(\llbracket r_1 \rrbracket, \llbracket r_1 \rrbracket_{2t}), \cdots, (\llbracket r_{t+1} \rrbracket, \llbracket r_{t+1} \rrbracket_{2t})$ optimistically. Finally, all parties try to reduce the degree of

$\llbracket ab \rrbracket_{2t}$ to get $\llbracket ab \rrbracket$, by (i) optimistically reconstructing $ab - r$ by interpolating the $2t$-degree shares $\llbracket ab - r \rrbracket_{2t}$ and (ii) adding $\llbracket r \rrbracket$ and $ab - r$ to get $\llbracket ab \rrbracket$. As such, the fast path can simply execute a sequence of such OptRanTriGen protocols, as long as the latest OptRanTriGen instance can output triples in time.

Algorithm 3 OptRanTriGen adapted from [11, 39, 66] ▷ Code of P_i

Input: System parameters
Output: Beaver_triples $= \{(\llbracket a_k \rrbracket, \llbracket b_k \rrbracket, \llbracket c_k \rrbracket)_{k=1}^{t+1}\}$
  // 1. Generate random double sharings
1: uniformly sample a randomness s_i
2: call $\mathsf{Share}(s_i, t)$ and $\mathsf{Share}(s_i, 2t)$, i.e., use t-degree and $2t$-degree plain Shamir secret sharing to share the same s_i
3: **wait** for all shares $(\llbracket s_j \rrbracket^i, \llbracket s_j \rrbracket_{2t}^i)$ sent from all n parties $\{P_j\}$:
4:    $(\llbracket r_1 \rrbracket^i, \cdots, \llbracket r_n \rrbracket^i) \leftarrow M(\llbracket s_1 \rrbracket^i, \cdots, \llbracket s_n \rrbracket^i)^T$
5:    $(\llbracket r_1 \rrbracket_{2t}^i, \cdots, \llbracket r_n \rrbracket_{2t}^i) \leftarrow M(\llbracket s_1 \rrbracket_{2t}^i, \cdots, \llbracket s_n \rrbracket_{2t}^i)^T$
6: **for** each $P_j \in \{P_{t+2}, \cdots, P_n\}$:
7:    P_i exclusively sends $(\llbracket r_j \rrbracket^i, \llbracket r_j \rrbracket_{2t}^i)$ to P_j
8: **if** $P_i \in \{P_{t+2}, \cdots, P_n\}$:
9:    **wait** for $(\llbracket r_i \rrbracket^j, \llbracket r_i \rrbracket_{2t}^j)$ sent from all n parties $\{P_j\}_{j \in [n]}$:
10:      **verify** that both secret shares have the correct degree
11:      **verify** that they can be decoded to get the same r_j
12:      **abort** if any verification fails, **continue** otherwise
  //The counterpart of line 12 in [11, 39, 66] reliably broadcasts a bit ("OK"/"ABORT") to cross-check the consistency of double sharing. Our fallback (Alg. 4) allows to remove the broadcasts.
13: rnd_dou_sha $\leftarrow \{(\llbracket r_1 \rrbracket^i, \llbracket r_1 \rrbracket_{2t}^i), \cdots, (\llbracket r_{t+1} \rrbracket^i, \llbracket r_{t+1} \rrbracket_{2t}^i)\}$
  // 2. Generate random sharings
14: repeat lines 1-12 twice without doing $2t$-degree sharing
15: if P_i does not abort in line 14, it obtains $2(t + 1)$ random shares rnd_sha $\leftarrow \{(\llbracket a_1 \rrbracket^i, \llbracket b_1 \rrbracket^i), \cdots, (\llbracket a_{t+1} \rrbracket^i, \llbracket b_{t+1} \rrbracket^i)\}$
  // 3. Perform degree reduction
16: **for** each $1 \le k \le t + 1$: compute $\llbracket c_k \rrbracket_{2t}^i = \llbracket a_k \rrbracket^i \cdot \llbracket b_k \rrbracket^i$
17: invoke $\mathsf{BatchRec}(\llbracket c_1 \rrbracket_{2t}^i - \llbracket r_1 \rrbracket_{2t}^i, \cdots, \llbracket c_{t+1} \rrbracket_{2t}^i - \llbracket r_{t+1} \rrbracket_{2t}^i)$ and wait for it returns $(c_1 - r_1, \cdots, c_{t+1} - r_{t+1})$
18: **for** each $1 \le k \le t + 1$: compute $\llbracket c_k \rrbracket^i = c_k - r_k + \llbracket r_k \rrbracket^i$
19: **return** Beaver_triples $= \{(\llbracket a_k \rrbracket^i, \llbracket b_k \rrbracket^i, \llbracket c_k \rrbracket^i)_{k=1}^{t+1}\}$

Final protocol of dual-mode triple generation. Clearly, the fast path built from OptRanTriGen cannot guarantee G.O.D. Considering the reconstruction of $2t$-degree shares in line 15 of Algorithm 3, it has to wait for all parties' shares to verify they are consistent w.r.t. a unique $2t$-degree polynomial. That means, if any party crashes, OptRanTriGen would stuck forever. Therefore, we introduce a fallback mechanism as described in Algorithm 4, which can securely restore robustness in the bad case that the fast path fails to progress in time or encounters inconsistent verification.

The subtlety of fallback in an asynchronous network is that different honest parties might quit the fast path with different progresses. Figure 5 illustrates how our fallback mechanism resolves this threat. We first let each party maintain a safe buffer to temporarily withhold the outputs of the latest two OptRanTriGen from immediate output (as their immediate
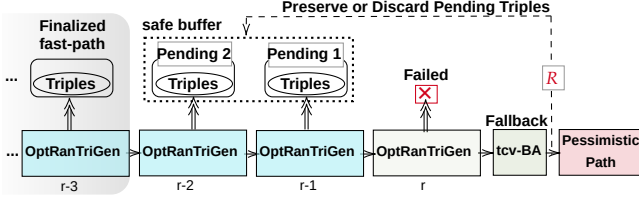
Figure 5: **Dual-mode triple generation with fallback**

output might cause disagreement if unexpectedly encountering fallback). Given the safe buffer, once a party P_i aborts in the r-th OptRanTriGen if detecting misbehavior or timeout, it can just invoke a two-consecutive-value Byzantine agreement (tcv-BA) with input $r-1$ (i.e., the index of the latest completed OptRanTriGen). Then, tcv-BA outputs R, and P_i reacts accordingly, to discard or preserve triples in its safe buffer. Finally, the honest parties would have totally consistent fast-path output after tcv-BA completes, and they can activate the pessimistic path by running AsyRanTriGen to recover G.O.D.

Security analysis of dual-mode triple generation. The security of Algorithm 4 can be summarized as follows.

Theorem 3 (Validity, Secrecy, and Liveness of Dual-mode Triple Generation). *Algorithm 4 securely realizes the desired properties of AMPC's offline phase of multiplication triple.*

Below, we briefly outline the security intuition. See Appendix G.3 of the full version for details.

- For validity, we first prove a lemma: if any honest party outputs in the r-th OptRanTriGen of fast path, then all honest parties already output triples up to the $(r$-1)-th OptRanTriGen. Then, for the validity and agreement of tcv-BA, all honest parties must obtain a common R during fallback, and R represents an OptRanTriGen instance where at least an honest party has outputted. Therefore, all honest parties can finalize consistent fast-path triples with the same progress according to R-1.

- For secrecy, it is straightforward as both fast and pessimistic paths have provable secrecy, and our sequential composition of them would inherit their secrecy.

- Liveness is immediate, as (i) everyone can quit from a failed fast path after timer expires, (ii) the termination of tcv-BA and AsyRanTriGen then ensure the pessimistic path to always progress even if in the worst case.

Algorithm 4 Dual-mode offline triple generation   ▷ Code for P_i

Input: System parameters
Output: A sequence of multiplication triple shares
     // Fast path
1: initialize a timer Δ that expires after τ
2: initialize safe buffer Pending_1 $= \emptyset$ and Pending_2 $= \emptyset$
3: **for** $r \in \{1, 2, 3, \cdots\}$:
4:     reset timer Δ and activate an OptRanTriGen$_r$ instance for r
5:     **if** OptRanTriGen$_r$ returns Beaver_triples$_r$ before Δ expires:
6:       **if** Pending_2 not empty: **output** Pending_2

7:       Pending_2 = Pending_1, Pending_1 = Beaver_triples$_r$
8:     **else** //If OptRanTriGen$_r$ either aborts or doesn't output in time
9:       activate tcv-BA with input $r-1$ and wait for it returns R
          //Preserve fast-path triples until the $(R$-1)-th OptRanTriGen
10:       **if** $R = r-2$: discard Pending_2 and Pending_1
11:       **if** $R = r-1$: **output** Pending_2 and discard Pending_1
12:       **if** $R = r$: **output** Pending_2 and Pending_1
13:       **break**
     // Pessimistic path
14: **while** true: run an AsyRanTriGen instance and **output** its result

---

Complexities. Our fast path prepares triples with amortized $O(n)$ communication, in the good case of a synchronous network without actual corruptions. In the worst case, our pessimistic path attains an amortized $O(n^2)$ per-triple overhead. Both paths achieve concrete improvements as well: (i) the fast path reduces the number of communication rounds by 3 compared to hbMPC; (ii) the pessimistic path reduces the number of shared secrets by a factor of 3 compared to GS23.

7   Towards Efficient AMPC-as-a-Service

Realizing Dumbo-MPC **as robust AMPCaaS.** Given our robust offline protocols, we can directly instantiate the robust offline phase of Dumbo-MPC using them. While for the online phase, it can mostly inherit the already performant and robust design from hbMPC, which consists of three main sub-phases: (i) function ordering, (ii) input solicitation and (iii) function evaluation. Here "function ordering" enables AMPC servers reach an agreement on the function to evaluate, "input solicitation" lets AMPC servers gather the private inputs from a sufficient number of clients (which usually is realized by a special variant of asynchronous Byzantine agreement known as asynchronous common subset, and can also be dedicatedly optimized as discussed in Appendix E of the full version [79]), and finally, "function evaluation" lets AMPC servers privately evaluate the function on the solicited private inputs.

Notably, each above online sub-phase can be realized via black-box invocation of standard techniques, and they are neither the bottleneck of efficiency nor the obstacles to robustness, so our primary focus is the design and implementation of a more efficient robust offline phase throughout the paper. For those reasons, we wouldn't repeat the online phase design here, and refer readers interested in its details to Appendix E of the full version [79].

Optimizing consensus component. Dumbo-MPC heavily relies on MVBA—a specific asynchronous Byzantine agreement protocol, to overcome network asynchrony by explicitly reaching consensus in both the online and offline phases. For efficiency of this critical component, we put forth the notion of optimistically terminable asynchronous MVBA (otMVBA) and give its generic construction. Our otMVBA construction is an MVBA that can deterministically and responsively terminate in 5 asynchronous rounds, under optimistic conditions like the network is synchronous and a certain party is honest.

As opposed, earlier MVBAs [4, 45, 56, 68] terminate in about 7 rounds after at least one invocation of randomized coin flipping, even if in the best conditions.[11] Our core idea to realize optimistic termination is forbidding the arbitrary choice of output during the pessimistic case, by enforcing a non-leader party to either (i) obtain a threshold signature attesting that no honest party would ever output in the good case, or (ii) wait for another mutually exclusive threshold signature fixing the good-case output. Readers interested in otMVBA can find sufficient details in Appendix H of the full version [79].

8   Implementation and Evaluations

We implemented Dumbo-MPC, especially its offline phase.[12] The triple generation protocol of GS23 [54] is also implemented for fair comparison. We finally evaluate and experimentally compare Dumbo-MPC, hbMPC [66] and GS23, in different network settings with varying system scales and different fault numbers. Note that we do not evaluate the incomparable protocols like DXKR23 [41] for generating double random sharing, as their pre-processed $2t$-degree secret shares could fail in reconstruction during the online phase, compromising either online robustness or optimal $n/3$ resilience.

8.1   Implementation and Experiment Setup

Implementation. All evaluated protocols are written in the same language of Python 3 as forks of the open-source implementations of hbACSS [80] and hbMPC [66]. The p2p channels utilize unauthenticated TCP sockets, and concurrent tasks are managed by Python's asyncio library. We use pairing friendly elliptic curve BLS12-381 for Dumbo-MPC, atop which Boldyreva's BLS threshold signature [18] and KZG polynomial commitment [62] are implemented. For GS23, we use secp256k1 curve to implement Pedersen commitments. All elliptic curve implementations are from gnark-crypto [22]. We choose a 256-bit prime field for Shamir secret sharing.

Experiment setup. We evaluate Dumbo-MPC and GS23 [54] using AWS EC2 c6a.8xlarge instances installing Ubuntu 20.04 LTS and equipped with 32 vCPUs and 64 GB memory.[13] We run tests with varying scales for $n = 4, 10, 22$, and 31 parties in the same AWS region at Virginia, which reflects a LAN deployment setting. As an affordable and reproducible WAN benchmarking approach, we utilize Linux TC tool to restrict the upload bandwidth of each instance to 500 Mbps and

---

[11] Ditto [49] once presented 2-chain VABA and claimed it as an MVBA with 5-round optimistic latency, but its recentest version withdraws the result.

[12] Note that the online phase is not efficiency bottleneck, and it can be mostly forked from hbMPC [66] with adaptions to its consensus components.

[13] Note that we choose the high-profile EC2 instance to accommodate large memory requirement instead of achieving parallelization speed-up, cf. our experiments conducted on different EC2 instances (see Appendix I.2 of the full version [79]), which reveals there is no performance gain through parallelization over extra CPUs.

set the network's round-trip time (RTT) to 150 ms, followed by re-running all tests in this simulated WAN setting.

8.2   Evaluation results

Choices of batch size. Dumbo-MPC has a critical parameter of batch size, which specifies the number of random secrets taken as input in each triple generation protocol. Clearly, the throughput of triples is closely related to the choice of batch size: A larger batch size might render a higher throughput, as a result of amortizing the fixed overheads, but an unnecessarily large batch might cause dramatic increment of latency. We evaluate Dumbo-MPC under varying batch sizes (1000, 5000, 10000, 15000 and 20000) in the LAN setting at different scales of n= 4, 10, 22 and 31. Figures 6 and 7 plot the trade-off between throughput and batch size in AsyRanTriGen and OptRanTriGen, respectively. Clearly, while the batch size increases, throughput starts to grow rapidly but soon becomes stable. In almost all scales, the throughput reaches a plateau after the batch size ≥ 5000, thus leading us to choose a fixed batch size of 5000 throughout all later experiments.
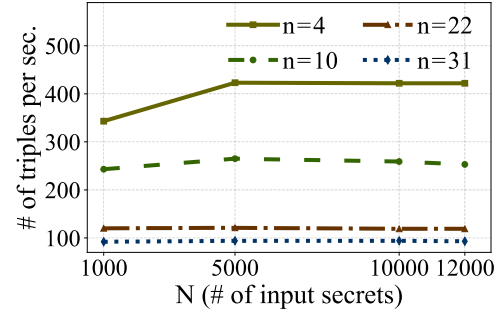
Figure 6: **Throughput v.s. batch in AsyRanTriGen.**
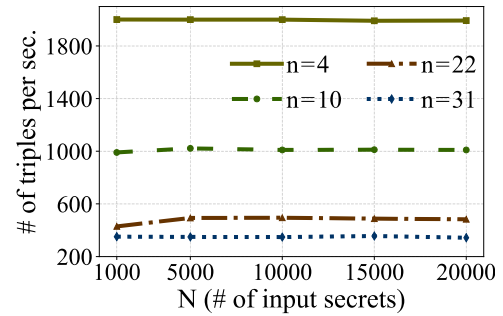
Figure 7: **Throughput v.s. batch in OptRanTriGen.**

Triple throughput. Figures 8 and 9 plot the throughput performance of Dumbo-MPC compared with GS23 and hbMPC, in the LAN and WAN settings, respectively. Though throughputs of all evaluated protocols decrease while the system scale is larger, AsyRanTriGen of Dumbo-MPC outperforms GS23 if $n \geq 10$ in LAN, and is always superior to GS23 in WAN. In particular, when $n = 31$, AsyRanTriGen realizes a throughput

of 94 triples/sec in LAN, 2X that of GS23. For our OptRanTri-Gen protocol, it can optimistically generate 349 triples/sec when $n = 31$ in the LAN setting, resulting in a throughput 6X of GS23 and about 10% larger than hbMPC. Moreover, restricting bandwidth to 500 Mbps and RTT to 150 ms only causes marginal decrement in triple throughput. When $n = 31$ in WAN, OptRanTriGen (resp. AsyRanTriGen) generates 339 (resp. 72) triples/sec, 7X (resp. 2X) that of GS23.
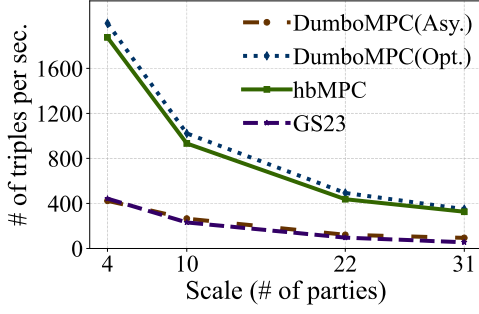

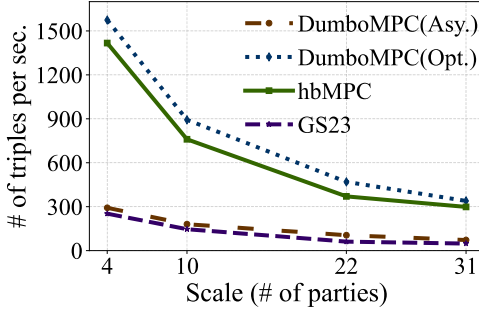Figure 8: **Throughput v.s. scale in the LAN setting.**

Figure 9: **Throughput v.s. scale in the WAN setting.**

Communication cost. Figure 10 reports the amortized per-node communication for each triple. As expected, OptRanTri-Gen and hbMPC have the least communication (i.e., constant per-node overhead about 10 KB for each triple). GS23 exhibits the worst asymptotic behavior and is concretely worse than AsyRanTriGen, since GS23 needs to share $3n$ secret per triple but AsyRanTriGen only shares n secrets per triple.
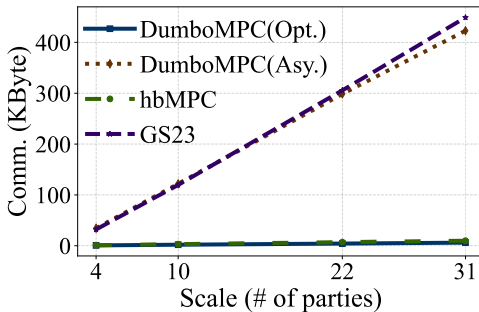

Figure 10: **Communication cost per node per triple.**

Performance with fallback. To understand the performance of the evaluated protocols in the bad case, we examine an execution that begins with optimistic conditions and later enters a bad case due to a malicious party. As Figure 11 plots, although both Dumbo-MPC and hbMPC can outperform GS23 during the good case, hbMPC grinds to a halt when a party misbehaves (as shown by the red region in Figure 11). Clearly, Dumbo-MPC achieves the best of both AsyRanTriGen and OptRanTriGen, harvesting efficiency in good case and simultaneously preserving robustness in all situations.
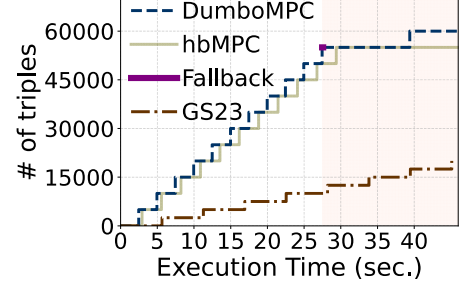

Figure 11: **Execution with fallback for $n = 4$ (in LAN).**

Additional tests. We defer more evaluation results like the impact of crash nodes and the pre-processing latency of specific tasks (auction and shuffling) to Appendix I of the full version [79].

9 Conclusion

We design Dumbo-MPC—a set of concretely efficient AMPC protocols in the classic offline-online paradigm, to circumvent the severe robustness-efficiency trade-off of existing AMPC protocols like GS23 and hbMPC. Dumbo-MPC is also the first implemented AMPC with all-phase G.O.D., featuring a novel dual-mode robust offline phase dedicated optimizations for each crucial sub-protocols. Extensive experiments showcase its promising performance in various applications.

Acknowledgments

Ethics Considerations

We have made every effort to address the ethical considerations necessary to ensure our work enhances security, privacy, and societal benefit while minimizing potential risks, during the course of designing and implementing our more efficient AMPC-as-a-Service protocol. This research deliverable is specifically crafted to improve the resilience and confidentiality of a fundamental privacy-preserving technique—secure multiparty computation, particularly in adversarial deployment environments where an attacker may influence the network and simultaneously compromise up to one-third of geologically distributed participants, with the goal of eliminating the need for a trusted third party to protect user privacy.

We recognize that, like many cryptographic technologies, our protocol could potentially be misused by malicious actors to conceal harmful activities in real-world applications. However, we are confident that the technical innovations and societal benefits of our protocol far outweigh these risks, as it provides a new solution to the significant challenge of mitigating large-scale privacy breaches of Internet peers.

Additionally, we are committed to responsibly disclosing any vulnerabilities that may arise, whether due to design flaws or implementation issues. Our research also emphasizes accessibility, inclusivity, and openness, ensuring that even organizations with limited resources and technical expertise can benefit from our work, advancing fairness in security and privacy research. Furthermore, our protocol adheres to all relevant legal and regulatory frameworks.

Through all these efforts, we aim to deliver secure, reliable, and ethically responsible secure multiparty computation protocols that are feasible for Internet deployment, with a focus on transparency and accessibility for the community.

Compliance with the Open Science Policy

Our entire codebase is packaged as a single self-contained open-source library and can be downloaded from the following link: https://github.com/dcy456/Dumbo-MPC. The permanent record of our code is also available at: https://zenodo.org/records/14678208.

References

[1] Ittai Abraham, Gilad Asharov, Shravani Patil, and Arpita Patra. Perfect asynchronous MPC with linear communication overhead. In *Proc. EUROCRYPT 2024*, pages 280–309, 2024.

[2] Ittai Abraham, Gilad Asharov, Arpita Patra, and Gilad Stern. Perfectly secure asynchronous agreement on a core set in constant expected time. *Cryptology ePrint Archive*, 2023.

[3] Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, and Gilad Stern. Bingo: Adaptively secure packed asynchronous verifiable secret sharing and asynchronous distributed key generation. pages 39–70, 2023.

[4] Ittai Abraham, Dahlia Malkhi, and Alexander Spiegelman. Asymptotically optimal validated asynchronous byzantine agreement. In *Proc. PODC 2019*, pages 337–346, 2019.

[5] Ittai Abraham, Benny Pinkas, and Avishay Yanai. Blinder–scalable, robust anonymous committed broadcast. In *Proc. CCS 2020*, pages 1233–1252, 2020.

[6] Gilad Asharov and Yehuda Lindell. A full proof of the bgw protocol for perfectly secure multiparty computation. *Journal of Cryptology*, 30(1):58–151, 2017.

[7] Assi Barak, Martin Hirt, Lior Koskas, and Yehuda Lindell. An end-to-end system for large scale p2p mpc-as-a-service and low-bandwidth mpc for weak participants. In *Proc. CCS 2018*, pages 695–712, 2018.

[8] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *Proc. CRYPTO 1991*, pages 420–432, 1992.

[9] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 503–513, 1990.

[10] Zuzana Beerliová-Trubíniová and Martin Hirt. Efficient multi-party computation with dispute control. In *Proc. TCC 2006*, pages 305–328, 2006.

[11] Zuzana Beerliová-Trubíniová and Martin Hirt. Perfectly-secure MPC with linear communication complexity. In *Proc. TCC 2008*, pages 213–230, 2008.

[12] M Ben-Or, Avi Wigderson, and S Goldwasser. Completeness theorems for noncryptographic fault-tolerant distributed computations. In *Proc. STOC 1988*, pages 1–10, 1988.

[13] Michael Ben-Or, Ran Canetti, and Oded Goldreich. Asynchronous secure computation. In *Proc. STOC 1993*, pages 52–61, 1993.

[14] Michael Ben-Or, Boaz Kelmer, and Tal Rabin. Asynchronous secure computations with optimal resilience. In *Proc. PODC 1994*, pages 183–192, 1994.

[15] Michael Ben-Or, Boaz Kelmer, and Tal Rabin. Asynchronous secure computations with optimal resilience (extended abstract). In *Proc. PODC 1994*, pages 183–192, 1994.

[16] Fabrice Benhamouda, Shai Halevi, and Tzipora Halevi. Supporting private data on hyperledger fabric with secure multiparty computation. *IBM Journal of Research and Development*, 63(2/3):3–1, 2019.

[17] Fabrice Benhamouda, Shai Halevi, Hugo Krawczyk, Yiping Ma, and Tal Rabin. SPRINT: High-throughput robust distributed schnorr signatures. In *Advances in Cryptology – EUROCRYPT 2024*, pages 62–91, 2024.

[18] Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In *Proc. PKC 2003*, pages 31–46, 2002.

[19] Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter M. R. Rasmussen, and Amit Sahai. Threshold cryptosystems from threshold fully homomorphic encryption. In *Proc. CRYPTO 2018*, pages 565–596, 2018.

[20] Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter MR Rasmussen, and Amit Sahai. Threshold cryptosystems from threshold fully homomorphic encryption. In *Proc. CRYPTO 2018*, pages 565–596, 2018.

[21] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-dnf formulas on ciphertexts. In *Proc. TCC 2005*, pages 325–341, 2005.

[22] Gautam Botrel, Thomas Piellard, Youssef El Housni, Arya Tabaie, Gus Gutoski, and Ivo Kubjas. Consensys/gnark-crypto: v0.11.2, 2023.

[23] Gabriel Bracha and Sam Toueg. Resilient consensus protocols. In *Proc. PODC 1983*, pages 12–26, 1983.

[24] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In *Annual cryptology conference*, pages 868–886, 2012.

[25] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 309–325, 2012.

[26] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. In *Proc. CRYPTO 2001*, pages 524–541, 2001.

[27] Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in constantinople: Practical asynchronous byzantine agreement using cryptography. *Journal of Cryptology*, 2005.

[28] Annick Chopard, Martin Hirt, and Chen-Da Liu-Zhang. On communication-efficient asynchronous mpc with adaptive security. In *Proc. TCC 2021*, pages 35–65, 2021.

[29] Ashish Choudhury. Optimally-resilient unconditionally-secure asynchronous multi-party computation revisited. Cryptology ePrint Archive, Paper 2020/906, 2020.

[30] Ashish Choudhury, Martin Hirt, and Arpita Patra. Asynchronous multiparty computation with linear communication complexity. In Yehuda Afek, editor, *Proc. DISC 2013*, pages 388–402, 2013.

[31] Ashish Choudhury and Arpita Patra. Optimally resilient asynchronous mpc with linear communication complexity. In *Proc. ICDCN 2015*, 2015.

[32] Ashish Choudhury and Arpita Patra. An efficient framework for unconditionally secure multiparty computation. *IEEE Transactions on Information Theory*, 63(1):428–468, 2017.

[33] Ashish Choudhury and Arpita Patra. On the communication efficiency of statistically secure asynchronous MPC with optimal resilience. *Journal of Cryptology*, 36(2):13, 2023.

[34] Ran Cohen. Asynchronous secure multiparty computation in constant time. In *Proc. PKC 2016*, pages 183–207. 2016.

[35] Sandro Coretti, Juan Garay, Martin Hirt, and Vassilis Zikas. Constant-round asynchronous multi-party computation based on one-way functions. In *Proc. ASIACRYPT 2016*, pages 998–1021, 2016.

[36] Ivan Damgård, Martin Geisler, Mikkel Krøigaard, and Jesper Buus Nielsen. Asynchronous multiparty computation: Theory and implementation. In *Proc. PKC 2009*, pages 160–179, 2009.

[37] Ivan Damgård, Yuval Ishai, Mikkel Krøigaard, Jesper Buus Nielsen, and Adam Smith. Scalable multiparty computation with nearly optimal work and resilience. In *Proc. CRYPTO 2008*, pages 241–261, 2008.

[38] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. Practical covertly secure mpc for dishonest majority – or: Breaking the spdz limits. In *Proc. ESORICS 2013*, pages 1–18, 2013.

[39] Ivan Damgård and Jesper Buus Nielsen. Scalable and unconditionally secure multiparty computation. In *Proc. CRYPTO 2007*, pages 572–590, 2007.

[40] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Proc. CRYPTO 2012*, pages 643–662, 2012.

[41] Sourav Das, Zhuolun Xiang, Lefteris Kokoris-Kogias, and Ling Ren. Practical asynchronous high-threshold distributed key generation and distributed polynomial sampling. In *Proc. USENIX Security 23*, pages 5359–5376, 2023.

[42] Sourav Das, Zhuolun Xiang, and Ling Ren. Asynchronous data dissemination and its applications. In *Proc. CCS 2021*, pages 2705–2721, 2021.

[43] Sourav Das, Zhuolun Xiang, and Ling Ren. Powers of tau in asynchrony, 2024.

[44] Sourav Das, Thomas Yurek, Zhuolun Xiang, Andrew Miller, Lefteris Kokoris-Kogias, and Ling Ren. Practical asynchronous distributed key generation. In *Proc. SP*, pages 2518–2534, 2022.

[45] Sisi Duan, Xin Wang, and Haibin Zhang. Fin: Practical signature-free asynchronous common subset in constant time. In *Proc. CCS 2023*, pages 815–829, 2023.

[46] Sisi Duan, Xin Wang, and Haibin Zhang. Practical signature-free asynchronous common subset in constant time. In *Proc. CCS 2023*, 2023.

[47] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *Cryptology ePrint Archive*, 2012.

[48] Zvi Galil, Stuart Haber, and Moti Yung. Cryptographic computation: Secure fault-tolerant protocols and the public-key model. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 135–155, 1987.

[49] Rati Gelashvili, Lefteris Kokoris-Kogias, Alberto Sonnino, Alexander Spiegelman, and Zhuolun Xiang. Jolteon and ditto: Network-adaptive efficient consensus with asynchronous fallback. In *Proc. FC 2022*, pages 296–315. Springer, 2022.

[50] Rosario Gennaro, Michael O Rabin, and Tal Rabin. Simplified vss and fast-track multiparty computations with applications to threshold cryptography. In *Proc. PODC 1998*, pages 101–111, 1998.

[51] Vipul Goyal, Yanyi Liu, and Yifan Song. Communication-efficient unconditional mpc with guaranteed output delivery. In *Proc. CRYPTO 2019*, pages 85–114, 2019.

[52] Vipul Goyal, Chen-Da Liu-Zhang, and Yifan Song. Towards achieving asynchronous MPC with linear communication and optimal resilience. In *Proc. CRYPTO 2024*, pages 170–206, 2024.

[53] Vipul Goyal, Yifan Song, and Chenzhi Zhu. Guaranteed output delivery comes free in honest majority mpc. In *Proc. CRYPTO 2020*, pages 618–646, 2020.

[54] Jens Groth and Victor Shoup. Design and analysis of a distributed ecdsa signing service. Cryptology ePrint Archive, Paper 2022/506, 2022.

[55] Jens Groth and Victor Shoup. Fast batched asynchronous distributed key generation. In *Proc. EUROCRYPT 2024*, pages 370–400, 2024.

[56] Bingyong Guo, Yuan Lu, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. Speeding dumbo: Pushing asynchronous bft closer to practice. 2022.

[57] Bingyong Guo, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. Dumbo: Faster asynchronous bft protocols. In *Proc. CCS 2020*, pages 803–818, 2020.

[58] Martin Hirt and Jesper Buus Nielsen. Robust multiparty computation with linear communication complexity. In *Proc. CRYPTO 2006*, pages 463–482, 2006.

[59] Martin Hirt, Jesper Buus Nielsen, and Bartosz Przydatek. Cryptographic asynchronous multi-party computation with optimal resilience. In *Proc. EUROCRYPT 2005*, pages 322–340, 2005.

[60] Martin Hirt, Jesper Buus Nielsen, and Bartosz Przydatek. Asynchronous multi-party computation with quadratic communication. In *Automata, Languages and Programming*, pages 473–485, 2008.

[61] Sanket Kanjalkar, Ye Zhang, Shreyas Gandlur, and Andrew Miller. Publicly auditable mpc-as-a-service with succinct verification and universal setup. In *2021 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 386–411, 2021.

[62] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *Proc. ASIACRYPT 2010*, pages 177–194, 2010.

[63] Marcel Keller. Mp-spdz: A versatile framework for multi-party computation. In *Proc. CCS 2020*, pages 1575–1590, 2020.

[64] Marcel Keller, Emmanuela Orsini, and Peter Scholl. Mascot: faster malicious arithmetic secure computation with oblivious transfer. In *Proc. CCS 2016*, pages 830–842, 2016.

[65] Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: making spdz great again. In *Proc. EUROCRYPT 2018*, pages 158–189, 2018.

[66] Donghang Lu, Thomas Yurek, Samarth Kulshreshtha, Rahul Govind, Aniket Kate, and Andrew Miller. Honeybadgermpc and asynchromix: Practical asynchronous mpc and its application to anonymous communication. In *Proc. CCS 2019*, pages 887–903, 2019.

[67] Yuan Lu, Zhenliang Lu, and Qiang Tang. Bolt-dumbo transformer: Asynchronous consensus as fast as the pipelined bft. In *Proc. CCS 2022*, pages 2159–2173, 2022.

[68] Yuan Lu, Zhenliang Lu, Qiang Tang, and Guiling Wang. Dumbo-mvba: Optimal multi-valued validated asynchronous byzantine agreement, revisited. In *Proc. PODC 2020*, pages 129–138, 2020.

[69] Deepak Maram, Harjasleen Malvai, Fan Zhang, Nerla Jean-Louis, Alexander Frolov, Tyler Kell, Tyrone Lobban, Christine Moy, Ari Juels, and Andrew Miller. Candid: Can-do decentralized identity with legacy compatibility, sybil-resistance, and accountability. In *Proc. SP 2021*, pages 1348–1366, 2021.

[70] Fabio Massacci, Chan Nam Ngo, Jing Nie, Daniele Venturi, and Julian Williams. Futuresmex: secure, distributed futures market exchange. In *Proc. SP 18*, pages 335–353, 2018.

[71] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of bft protocols. In *Proc. CCS 2016*, pages 31–42, 2016.

[72] Christian Mouchet, Bertrand Elliott, and Hubaux Jean Pierre. An efficient threshold access-structure for rlwe-based multiparty homomorphic encryption. *Journal of Cryptology*, 36(10):1–20, 2023.

[73] Arpita Patra, Ashish Choudhury, and C Pandu Rangan. Efficient asynchronous verifiable secret sharing and multiparty computation. *Journal of Cryptology*, 28:49–109, 2015.

[74] Arpita Patra, Ashish Choudhury, and C. Pandu Rangan. Efficient asynchronous multiparty computation with optimal resilience. Cryptology ePrint Archive, Paper 2008/425, 2008.

[75] B Prabhu, K Srinathan, and C Pandu Rangan. Asynchronous unconditionally secure computation: An efficiency improvement. In *Proc. INDOCRYPT 2002*, pages 93–107, 2002.

[76] Marc Rivinius, Pascal Reisert, Daniel Rausch, and Ralf Küsters. Publicly accountable robust multi-party computation. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 2430–2449, 2022.

[77] Victor Shoup and Nigel P Smart. Lightweight asynchronous verifiable secret sharing with optimal resilience. *Journal of Cryptology*, 37(3):27, 2024.

[78] K Srinathan and C Pandu Rangan. Efficient asynchronous secure multiparty distributed computation. In *Proc. INDOCRYPT 2000*, pages 117–129, 2000.

[79] Yuan Su, Yuan Lu, Jiliang Li, Yuyi Wang, Chengyi Dong, and Qiang Tang. Dumbo-MPC: Efficient fully asynchronous MPC with optimal resilience. Cryptology ePrint Archive, Paper 2024/1705, 2024. URL: https://eprint.iacr.org/2024/1705.

[80] Yurek Thomas, Luo Licheng, Fairoze Jaiden, Kate Aniket, and Miller Andrew. hbacss: How to robustly share many secrets. In *Proc. NDSS 2022*, pages 1187–1201, 2022.

[81] Antoine Urban and Matthieu Rambaud. Robust multi-party computation from threshold encryption based on rlwe. *Cryptology ePrint Archive*, 2024.

[82] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Global-scale secure multiparty computation. In *Proc. CCS 2017*, pages 39–56, 2017.

[83] Thomas Yurek, Zhuolun Xiang, Yu Xia, and Andrew Miller. Long live the honey badger: Robust asynchronous {DPSS} and its applications. In *Proc. USENIX Security 23*, pages 5413–5430, 2023.

[84] Haibin Zhang and Sisi Duan. Pace: Fully parallelizable bft from reproposable byzantine agreement. In *Proc. CCS 2022*, pages 3151–3164, 2022.

[85] Guy Zyskind, Oz Nathan, and Alex Pentland. Enigma: Decentralized computation platform with guaranteed privacy. *arXiv preprint arXiv:1506.03471*, 2015.

A   Additional Related Studies and Alternative Design Choices

A.1   Additional related work

MPC has attracted a great amount of attentions since its formulation in 1980's, and it has been a subject involving a huge

line of studies. Here we go over a few relevant results and discuss their limitations in the context of realizing more efficient fully asynchronous MPC.

MPC protocols based on additive secret sharing, such as SPDZ [38] and EMP [82] and its descendants [63], can tolerate dishonest majority with resilience $t < n$. However, they would be aborted if there is any corrupted party, and thus might fail to realize G.O.D. The lack of G.O.D. (or more general, the loss of fairness) is inherent for any dishonest majority MPC protocols without extra setups.

Synchronous MPC in the honest majority setting has been extensively studied [11, 48, 51, 53, 58], and achieved $O(n)$ per-gate communication complexity, but their robustness and even privacy might be violated in an asynchronous network. In particular, dispute control [10], that is restarting the computation after detecting and eliminating faulty parties, is used to handle faulty parties for $t < n/3$ [11, 39] and $t < n/2$ [39, 53, 58] to construct secure MPC protocols in synchronous network. However, such protocols depending on the ability to time out participated parties are vulnerable to unresponsive attack [66]. As such, if t honest parties are temporarily isolated from the network due to asynchrony, the honest nodes would be removed from the system, and the remaining malicious parties may jeopardize the privacy of (clients') private inputs. HyperMPC [7] is designed to adapt to low bandwidth MPC based on [11] in the $n/3$ setting, and fails to guarantee output as it works with $2t$-shares both in the offline and online phase. In the offline phase, random double sharing of security with abort is needed to generate double shares (e.g., degree-t shares and a degree-$2t$ shares of some random secrets). During online phase, parties have to reconstruct $2t$-shares, which is infeasible to provide G.O.D. in the asynchronous setting with $n/3$.

In the asynchronous setting, MPC protocols usually employ asynchronous agreement protocols (e.g., asynchronous common subset [15, 45, 57, 71, 84]) to evaluate the function with only the $n - t$ fastest parties' inputs. The topic was initially studied in the setting of computationally-unbounded adversaries [13, 14, 30, 32, 73, 75, 78]. Perfectly secure asynchronous MPC without any error probability recently was also designed for ($t < n/4$) with linear per-gate communication overhead [1]. Unconditionally secure asynchronous MPC protocols can also be designed to realize optimal $t < n/3$ tolerance (in the presence of negligible small error probability). In this setting, the first design [15] incurs $O(n^{11})$ per-gate communication complexity, and later was improved to $O(n^5)$ [74] and $O(n^4)$ [29, 33]. Very recently, Goyal et al. [52] proposed the first unconditionally secure AMPC with linear per-gate communication and optimal resilience of $t < n/3$, but unfortunately, it has a prohibitive $O(n^{14})$ circuit-independent communication overhead (which might be not surprisingly as the stringent setting of unconditional information-theory setting brings extra design overhead). Their AMPC consists of two processes: process 1 needs to share $\frac{144N}{\varepsilon^2 \cdot n} + \frac{54}{\varepsilon^2 \cdot n}$ se-

crets to generate one triple, and process 2 requires to share $\frac{108(2N+3)}{\varepsilon^3 \cdot n} + \frac{27}{\varepsilon^2 \cdot n} + \frac{N}{n}$, where N is the batch size, n is the number of parties, and ε is suggested as 0.1. Setting $N = O(n)$, generating one triple necessitates the sharing of at least 14400 secrets in process 1, and at least 216000 secrets in process 2.

For computationally secure AMPC using cryptography, several protocols are proposed based on additive/somewhat threshold homomorphic encryption [28, 31, 60], where arithmetic operations are performed on ciphertexts in the online phase. In [31], threshold somewhat homomorphic encryption (e.g., BGN cryptosystem [21]) is used as a primitive to construct encrypted MPC protocol, which might take expected $\tilde{O}(2^{\sqrt{w}})$ time to decrypt ciphertext using Pollard's lambda method (where w is the bit-length of secret share field), likely causing its implementation only suitable for Boolean circuits (or arithmetic circuits defined over very small fields). hbMPC [66] does not require any homomorphic encryption, and it is proposed to guarantee output in the online phase but only provides security with abort in the offline phase (since its offline requires to reconstruct $2t$-shares).

A recent work of Das et al. [41] presents an efficient asynchronous random double sharing (ARDS) protocol, but the result alone fails to robustly reconstruct $2t$-shares while evaluating a multiplication gate in the online phase for $t < n/3$. Very recently, a couple of studies [54, 77] achieve guaranteed output delivery in asynchrony, but with cubic per-gate communication overhead in the worst case.

Other theoretic asynchronous MPC protocols feature a constant-round online phase independent of the circuit depth, making use of different underlying cryptographic primitives, such as FHE [34] and pseudo-random generator [35]. But they either exhibit a large communication overhead that is quadratic in n and linear in circuit size (to privately compute a distributed version of garbled circuit through another AMPC protocol like ours) [35], or/and suffer from large online computational cost like expensive FHE self-ootstrappings that are linear to the circuit's multiplicative depth [34].

A.2 Efficiency issues of alternative designs

Noticeably, there indeed exist a few alternative design choices of asynchronous triple generation with asymptotic complexities that are similar to or even better than ours, but their concrete performance could be problematic. Here we briefly explain the reasons of their inferior performance.

Possible instantiations of Goyal et al. [52]. The AMPC protocol of Goyal et al. [52] achieves a communication cost of $O(|C|n + Dn^2 + n^6\kappa + n^7)$ plus $O(n^2)$ invocations of ACSS to share $O(|C|)$ degree-t Shamir sharings, where $|C|$ is circuit size and D is circuit depth. Though Goyal et al. [52] originally focus on the information theoretic setting, we nevertheless can instantiate the protocol by more efficient computationally-secure components, using the best-possible (computationally-secure) ACSS [3, 80] and coin flipping [27]. However, for

concrete efficiency, the protocol of Goyal et al. [52] needs to verifiably share hundreds or thousands of secrets for each triple, if using their suggested security parameters. Though we require n secrets to be shared for each triple (which is asymptotically worse), our approach is still concretely more efficient for typical system scales like several dozens of parties.

Possible instantiations of Choudhury et al. [33]. Noticeably, if we employ the state-of-the-art computationally-secure ACSS, such as hbACSS and Bingo, to instantiate the recent (statistically-secure) AMPC framework from Choudhury et al. [33], an AMPC with quadratic per-gate communication overhead can also be realized, but its concrete efficiency is significantly worse than our design, as it requires $6n$ secrets to be shared for generating each triple, which is six times larger than ours and three times larger than GS23. Shoup and Smart [77] also observe that a happy path can be embedded to the AMPC framework of Choudhury et al. [33] by (optimistically) waiting for messages sent from more than $n - t$ parties, but this happy path has an execution flow same to the original fully asynchronous protocol, which involves too many redundant communication rounds to circumvent asynchrony, making it concretely less efficient than our dedicated fast path particularly tailored for the good case.

AMPC from tSHE/tFHE. In the Shamir-based tFHE/tSHE scheme, smudging noise could be amplified by Lagrange coefficients during reconstruction, resulting in a modulus increase on the order of $O(n!^3)$ [19]. To mitigate this modulus growth, binary coefficients have been utilized in constructing tFHE/tSHE schemes, but such approach incurs significant space overhead, as the size of each secret key share grows to at least $O(n^{4.2})$. Some recent works [72, 76] employ a thresholdizer that uses additively shared secret keys to construct tFHE/tSHE where each additive key share is re-shared leveraging Shamir secret sharing. But this thresholdizer essentially requires all nodes to remain online to perform decryption, otherwise when some party is not responsive due to trigger some timeout, it would invoke the reconstruction of Shamir secret sharing to publicly open this lost additive share, making it clearly unsuitable for asynchronous settings (because an asynchronous adversary can force to recover all honest parties' additive key shares). It is also well known that we can also use pre-shared noises to address the parameter explosion caused by Lagrange interpolation in the Shamir-based tFHE/tSHE scheme [81]. However, these noises must correctly be shared within a specific range to follow an expected distribution, whose efficient distributed generation in the asynchronous network is another challenging problem to study.